Lightning Data Transport I/O Link Protocol Specification

Revision 1.01a

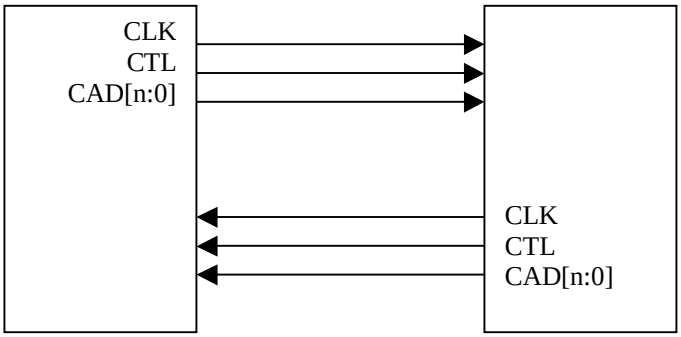
~~AMD Company Confidential~~

Revision History

Revision	Description	Date
0.18	Add 2- & 4-bit link definition. Tweak subtractive decode appendix	19-Jan-00
0.19	Add Dword based flow control. Add LDTStop and finish of SysMgt Editorial cleanup & clarification	24-Feb-00
0.20	Change LDTStop to active low Simplify LDTStop disconnect sequence Tweak SysMgt encoding & delete thermal alert encoding InitializationComplete false==EndofChain true (init req't) Define link frequency initialization and selection Add detail to host ordering requirements Lots of editorial cleanup	6-Mar-00
0.21	Lots of editorial cleanup. SysMgt packet: added VID/FID, WBINVD, INVD codes Redefinedsync mode and async clocking modes	8-Apr-00
1.0	Make consistent use of "command" vs "request". Added isoc traffic considerations. Simpler ISA deadlock avoidance – PREQ deleted. Modified LinkFail and InitializationComplete definition. Simplified and clarified Link Freq control registers. Split FERR and SMIACK into two SysMgt packets. LDTSTOP# disconnect can occur in middle of data packet Byte Write with Count==0 is illegal (but for intr&sysmgt) Redefined clock modes: sync, pseudo-sync & async Tweaked LDTSTOP# reconnect sequence Lots of editorial cleanup.	10-Jun-00
1.01	Made explicit that RESET# & PWROK are always inputs. Removed nonposted broadcast Added Atomic RMW requests. Interrupt packet contains 32 destination bits (was 8) ISA Enable & VGA Enable became optional for bridges. Secondary Bus Reset made required for bridges. Interrupt Line & Pin registers: same as PCI Added LinkFreqCap register to capability block Reserved x86-specific code points in SysMgt packet Added (optional) ISOC mode and moved "Isochronous Support" chapter (5) to appendix D Added config bit for state of link under LDTSTOP# Modified LDTSTOP# transmitter rules Added LinkWidthIn/Out code point for unconnected link Added x86 platform appendix Made CRC Testing (Diag) Mode optional Added config field for Host DeviceID Made double-hosted chain support required for tunnels Added Vector to Startup Interrupt Messages Editorial cleanup, clarifications	20-Oct-00
1.01a	Fix location of Feature Capability Register Editorial cleanup, clarifications	11-Dec-00

1 Overview

This document describes the lightning data transport (LDT) I/O link. LDT is a packet-based link implemented on two unidirectional sets of wires. The link is packet-based, nominally point-to-point, and connects exactly two devices. Devices can have multiple LDT links, allowing the construction of larger LDT fabrics.



LDT is used as an I/O channel, connecting chains of LDT I/O devices and bridges to a host system. The interface from the host to the LDT chain(s) is called the host bridge.

1.1 LDT Terminology

- A ***cycle*** is a period of time defined by the period of the clock (rising edge of CLK to rising edge of CLK).
- A ***bit time*** is half of a clock period in duration—two data bits are transmitted on each wire per cycle.
- A ***node*** is a physical entity that connects to one end of an LDT link.
- A ***packet*** is a series of bit times and forms the basis of communication between two nodes.
- A ***transaction*** is a sequence of packets that are exchanged between two or more nodes in the system and that result in a transfer of information.
- A ***source*** is the node that starts a transaction.
- A ***target*** is the node that ultimately services the transaction on behalf of the source. Note that there may be intermediary nodes between the source and the target.
- A ***unit*** or ***function*** is a logical entity within a node that can act as a source or destination of transactions. Functions are useful for describing the transaction ordering rules for LDT.
- A ***doubleword (DW)*** is four bytes.
- A ***quadword (QW)*** is eight bytes.
- ***I/O Streams*** are groups of transactions that can be treated independently by the routing fabric with respect to ordering considerations.

2 LDT Signals

Signal	Width	Description
CAD	2, 4, 8, 16, or 32	**C**ommand / **A**ddress / **D**ata. Carries LDT requests, responses, addresses and data. CAD width can be different in each direction.
CTL	1	When asserted, the CAD signals carry a control packet. When deasserted the CAD signals carry data.
CLK	1, 2, or 4	Clocks for the above signals. Each byte of CAD has a separate clock signal. Regardless of the width of the link, CTL is clocked by the same CLK as CAD[0].

The signals given in the table constitute a single unidirectional connection between two nodes. A full link requires a connection in each direction. However, the connections need not be the same width in each direction.

LDT links wider than 8 bits are built by ganging multiple 8-bit links in parallel to form either 16-bit or 32-bit links. Links wider than 8 bits have one clock per byte, but still only one CTL bit for the whole link. The forwarded clock for the CTL signal is the clock for the least-significant byte.

In addition to the link signals, all LDT devices require the following reset/initialization input pins:

Signal	Width	Description
PWROK	1	Power and clocks are stable.
RESET#	1	Reset the LDT chain

All devices in a given LDT chain receive the same PWROK and RESET# signals. In some implementations, these signals may be open drain wired-OR signals, thus allowing multiple sources (possibly including host bridges) to initiate a reset sequence. LDT devices must sample PWROK and RESET# as inputs, and may optionally drive these signals as outputs. These signals control the power-up and reset sequence for their LDT links, and may or may not also control the power-up and reset sequence for other logic within the device – this is device-specific. See section 4.1.1 for a description of these signals in the context of systems with multiple LDT chains. See section 12 for information on reset sequencing.

LDT devices deployed in PC-class or other systems requiring power management may include the following signals, which are used during the sequencing of system activities such as power savings state transitions. These signals are open-drain wired-OR, allowing multiple sources to request link disconnection and reconnection.

Signal	Width	Description
LDTSTOP#	1	Used to enable and disable links during system state transitions.
LDTREQ#	1	Used to request reenabling links for normal operation.

3 Packet Definition

This section describes the packet definition for the LDT link. LDT supports link widths of 2, 4, 8, 16, and 32 bits. All figures shown in this section assume an 8-bit wide link.

The packet structure for 16-bit and 32-bit links can be derived from the 8-bit link packet structure by combining the fields within adjacent bit times. Some examples:

$BT1_{16}[15:0] = BT2_{8}[7:0] \,||\, BT1_{8}[7:0]$

$BT1_{32}[31:0] = BT4_{8}[7:0] \,||\, BT3_{8}[7:0] \,||\, BT2_{8}[7:0] \,||\, BT1_{8}[7:0]$

Where BTN_m represents the Nth bit time within a packet for an LDT link of width m and "||" represents concatenation.

Since all packets are multiples of 4 bytes long, packet boundaries always fall on bit-time boundaries in the 16-bit and 32-bit case, as well as the 8-bit case.

The packet structure for 2-bit and 4-bit links can be derived from the 8-bit link packet structure by splitting the 8-bit link bit times into adjacent bit times. Some examples:

$BT1_{2}[1:0] = BT1_{8}[1:0]$
$BT2_{2}[1:0] = BT1_{8}[3:2]$
$BT3_{2}[1:0] = BT1_{8}[5:4]$

$BT4_2[1:0] = BT1_8[7:6]$

$BT1_4[3:0] = BT1_8[3:0]$
$BT2_4[3:0] = BT1_8[7:4]$

3.1 Use of the CTL Signal

LDT consists of control packets and data packets, distinguished by the use of the CTL signal. Link transmitters assert CTL during all bit times of control packets, and deassert it during data packets. The purpose of the CTL signal is to allow control packets to be inserted in the middle of long data packets, and to stall in the middle of control packets.

Here are the rules that govern packet transmission.

1. CTL is asserted through all bit times of a control packet.
2. CTL may be deasserted on 4-byte boundaries within a control packet, to insert stalls. No valid information is transmitted on the link during the stall. When CTL is reasserted, the interrupted control packet continues from where it left off.
3. CTL is deasserted through all bit times of a data packet.
4. CTL may be asserted on 4-byte boundaries within a data packet to insert a control packet. Control packets inserted into data packets must not themselves have an associated data packet. CTL may still be deasserted on 4-byte boundaries within the control packet to cause stalls. When CTL is deasserted at the conclusion of a control packet, data transfer continues from the point where it left off.
5. Write request and read response packets always have an associated data packet. The data packet might not immediately follow the last bit time of its associated control packet, as other control packets may be inserted before the data packet. However, because these control packets cannot have associated data, there can only be one data transfer outstanding.
6. The order of operations on the link is determined by the order of the control packets. The fact that data transfer for a control packet may be delayed does not affect how it is ordered.
7. The bit time immediately following the last bit-time of a data packet is always the start of a control packet (CTL must be asserted).
8. CTL may only be asserted or deasserted on a 4 byte boundary.
9. CTL may only be deasserted when data transfer due to a previously transmitted control packet is being sent, or in the middle of a control packet.

3.2 Packet Structure

This section defines the basic control and data packet types, and shows the position of the fields that are common to all the control packet types. All packets are multiples of four bytes long.

3.2.1 Control Packets

Control packets consist of four or eight bytes. This section shows the basic structure of each of these control packet forms.

In the diagrams that follow, the unlabelled packet fields are command-specific. Some common control packet fields are as follows:

- ***Cmd[5:0]*** is the command field which defines the packet type.
- ***UnitID[4:0]*** serves to identify participants in a transaction. Since all packets are transferred either to or from the host bridge at the end of the fabric, either the source or destination node is implied. The value zero is reserved for the UnitID of the host bridge. See section 4.2 for more details on the use of UnitID. Nodes with multiple logical I/O streams can own multiple UnitID values.
- ***Bridge*** indicates that this response packet was placed onto the link by the host bridge, and it is used to distinguish responses travelling upstream from responses travelling downstream. In the case of two host bridges sending packets to each other on a double-ended chain, the target host bridge appears to

the requesting host bridge as an LDT slave device. Therefore, the bridge bit will be clear on responses to requests issued from the far host bridge.

- ***SeqId[3:0]*** is used to tag groups of requests which were issued as part of an ordered sequence by a device and must be strongly ordered within a virtual channel. All requests within the same I/O stream and virtual channel that have matching nonzero seqId fields must have their ordering maintained. The seqId value of 0x0 is reserved to mean that a transaction is not part of a sequence. Transactions with this value have no sequence ordering restrictions, although they may be ordered for other reasons as described in section 6.
- ***PassPW*** indicates that that this packet is allowed to pass packets in the posted request channel in the same I/O stream. Otherwise, this packet must stay ordered behind them.
- ***SrcTag[4:0]*** is a transaction tag which is used to uniquely identify all transactions in progress initiated by a single requester. Each UnitID can have up to 32 transactions in progress at one time. The concatenation of source UnitID and SrcTag serves to uniquely identify nonposted requests. The SrcTag field is not relevant for posted requests, and is reserved.
- ***Addr[39:2]*** represents the doubleword address accessed by the request. Not all address bits are included in all request types. Where finer granularity is required, byte masks are used.

Reserved fields in command packets must always be driven to 0 by transmitters, and must be assumed to be undefined by receivers.

3.2.1.1 Info Packet

Info packets are always four bytes long. They are used for nearest neighbor communication between nodes, and so exist at the lowest level of the protocol. They are not routed within the fabric, and they require no buffering in the nodes. They are not flow-controlled, and they can always be accepted by their destination.

Bit Time	7	6	5	4	3	2	1	0
0	*Command-Specific*		Cmd[5:0]					
1	*Command-Specific*							
2	*Command-Specific*							
3	*Command-Specific*							

3.2.1.2 Request Packet

Request packets are either four or eight bytes long, depending upon whether the request has an associated address. The diagram below shows a request packet with an address. Four-byte request packets do not contain the address field.

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	*Command-Specific*							
3	*Command-Specific*							
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

3.2.1.3 Response Packet

Response packets are always four bytes long.

Bit Time	7	6	5	4	3	2	1	0
0	*Command-Specific*		Cmd[5:0]					
1	PassPW	Bridge	*Cmd-Specific*	UnitID[4:0]				
2	Rsv		Error	*Command-Specific*				
3	Rsv		NXA	Rsv			*Command-Specific*	

The Error bit is present in all responses, and is used to indicate that the issued request could not be completed. If the error bit is set, the non-existent address (NXA) bit is valid. If it is set, that means that the request could not be completed because no agent on the chain accepted the request. If the bit is clear, it means that the request packet reached its addressed target, but could not be completed by the device.

3.2.1.4 Command Field Encodings

The command field is valid for all control packets.

Code	VChan	Command	Comments/Options	Packet Type
000000	-	Nop	Null packet. Contains flow control info	Info
000001		Reserved-HOST		
000010	NPC	Flush	Flush posted writes	Request
000011 0001xx		Reserved-HOST		
x01xxx x01xxx	NPC or PC (bit 5)	Wr (sized)	Write Request [5] defines whether request is posted. 0: Non-Posted 1: Posted [2] defines the data length. 0: Byte 1: Doubleword [1] defines bandwidth/latency requirements. 0: Normal 1: Isochronous [0] indicates whether access requires host cache coherence (reserved if access is not to host memory). 0: Noncoherent 1: Coherent	Req/Addr/Data

Code	VChan	Command	Comments/Options	Packet Type
01xxxx 01xxxx	NPC	Rd (sized)	Read Requests [3] defines ordering requirements for response. 0: Response may not pass posted requests 1: Response may pass posted requests [2] defines the data length. 0: Byte 1: Doubleword [1] defines bandwidth/latency requirements. 0: Normal 1: Isochronous [0] indicates whether access requires host cache coherence (reserved if access is not to host memory). 0: Noncoherent 1: Coherent	Req/Address
100xxx		Reserved-I/O		
110000	R	RdResponse	Read Response.	Resp/Data
110001 110010		Reserved-HOST		
110011	R	TgtDone	Tell source of request that target is done.	Response
11010x		Reserved-HOST		
11011x		Reserved-I/O		
11100x		Reserved-HOST		
111010	PC	Broadcast	Broadcast Message.	Req/Address
111011		Reserved-IO		
111100	PC	Fence	Fence for posted requests.	Request
111101	NPC	Atomic-RMW	Atomic Read-Modify-Write	Req/Addr/Data
111110		Reserved-I/O		
111111	-	Sync/Error	Link Synchronization & Error Packet	Info

The fields in the table are as follows:

- ***Code*** is the 6 bit command encoding in each packet.
- ***VChan*** indicates the virtual channel that the packet travels in. Info packets are only used for single-link communication and don't use buffer space, and thus are not in a virtual channel. See sections 4.7 and 4.8 for more information. **PC** = Posted Command (Request), **NPC** = Non-Posted Command (Request), **R** = Response.
- ***Command*** is the mnemonic used to represent the command.
- ***Comments/Options*** gives a short description of the command, and enumerates any option bits within the ***Code*** field.
- ***Packet Type*** indicates the type of packet(s) that this command uses.

In the above table **Reserved-I/O** identifies code points that are reserved for future use in the LDT I/O Link Specification. **Reserved-HOST** identifies code points that may be used in a host-specific protocol, and will not be used to implement future features in the LDT I/O Link Specification.

The action taken by a device in response to receiving a packet with a reserved command code is UNDEFINED.

3.2.2 Data Packet

Data packets contain the data payload for transactions. Data packets follow write request and read response packets. Data packets range in length from four to 64 bytes, in multiples of four bytes (one doubleword). Within a doubleword, data bytes appear in their natural byte lanes. For transfers of less than a full doubleword, the data is padded with undefined bytes to achieve this byte-lane alignment.

This example shows an eight-byte data packet.

Bit Time	7	6	5	4	3	2	1	0
0	Data[7:0]							
1	Data[15:8]							
2	Data[23:16]							
3	Data[31:24]							
4	Data[39:32]							
5	Data[47:40]							
6	Data[55:48]							
7	Data[63:56]							

The data packet for a sized read response is arranged with the lowest addressed doubleword returned first and the remainder of the addressed data returned in ascending address order by doubleword. Sized read responses can contain any number of contiguous doublewords within a 64-byte aligned block, although for sized byte reads not all bytes are guaranteed to be valid. The data cannot wrap from the most significant Dword in the aligned 64-byte block to the least significant Dword in the block.

Sized doubleword writes work in the same way as sized doubleword read responses, and can contain anywhere from one to 16 doublewords in ascending address order.

Sized byte writes transmit one doubleword's worth of masks first, followed by from one to eight doublewords of data in ascending address order. Mask[0] corresponds to Data[7:0], Mask[1] to Data[15:8], and so on. Thirty-two mask bits are always transmitted, regardless of the amount of data. All-zero byte masks are permitted. However, if any byte masks are nonzero, there must be at least one set mask bit associated with the first doubleword of data. Interrupt and system management messages, which are composed of byte write packets to predefined address ranges, are the only byte write packets which do not require at least one doubleword of data.

Bit Time	7	6	5	4	3	2	1	0
0	Mask[7:0]							
1	Mask[15:8]							
2	Mask[23:16]							
3	Mask[31:24]							
4	Data[7:0]							
5	Data[15:8]							
6	Data[23:16]							
7	Data[31:24]							
8+	Packet may contain up to 8 doublewords of data							

4 Fabric Operation

The LDT link is a pipelined, split-transaction interconnect in which transactions are tagged by the source, and responses can return to the source out of order. This section outlines the basic operation of the link.

4.1 Topology

LDT I/O fabrics are implemented as one or more daisy chains of LDT devices, with a bridge to the host system at one end. Devices can implement either one or two links. A dual-link device that is not a bridge is called a tunnel. Single-link devices must always sit on the end of the chain, so only one single-link device is possible in a chain. Direct peer-to-peer communication between devices in the chain is not allowed. All packets (except for info packets) travel between one device and the host bridge. This means that at a high level, the fabric appears as a group of devices directly connected to the host bridge, but not to each other. Packets flowing into the fabric from a host bridge are said to be flowing downstream. Packets flowing to a host bridge from an LDT device are said to be flowing upstream.

A single LDT chain contains no LDT to LDT bridge devices. It may contain native LDT peripheral devices (like an ethernet controller), and may also contain bridges to other interconnects (like PCI). A LDT chain is terminated at one or both ends by a bridge. In the simplest topology, a LDT chain connects to the host bridge at one end and has no connection at the other end.

A LDT tree contains one or more LDT to LDT bridge devices. A LDT bridge device has a primary link, that being the upstream link in the direction of the host, and one or more secondary links. Each LDT chain that connects to a bridge's secondary link is assigned a unique bus number – see section 7.2.3 for details. The LDT to LDT bridge device operates as a host bridge for devices on its secondary chain. In addition to its secondary links, a LDT bridge device may have a downstream link that is associated with the same bus number as the bridge's primary link. The root of the LDT tree connects to the host.

The host can contain multiple bridges, each supporting either a single LDT chain or a tree of LDT chains.

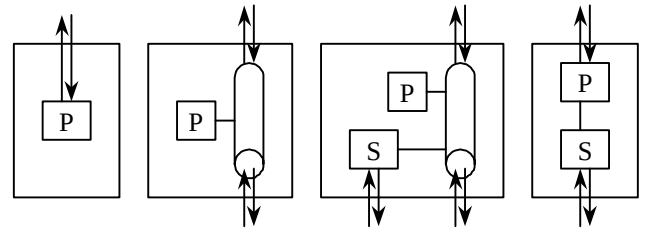


Figure 4-1 Example LDT Device Configurations

In these figures, "P" indicates a primary interface block and "S" indicates a secondary interface block. See section 7.5 for details.

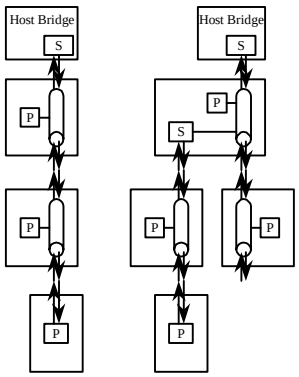


Figure 4-2 Example LDT Topologies

For convenience in integrating multiple functions onto a single chip, or to allow parallelism between independent request streams, individual LDT devices can use multiple UnitID values. There is no specific limit on the number of physical devices. However, there are only 31 UnitIDs available to each LDT chain. No combination of devices that exceeds 31 UnitIDs may be connected to a single chain.

Physically, a chain can be connected to a host bridge at each end, as long as the chain contains no single-link devices. This may be useful to provide another path to I/O devices in the event of a host bridge or link failure, or to allow sharing of I/O devices between independent hosts to implement clustering. One bridge is designated the master bridge for the shared chain, while the other will be the slave bridge. This designation must be made by hardware before the fabric is reset. The method of doing so is beyond the scope of this specification.

Logically, a double-ended chain appears as two distinct daisy-chains, each attached to only one host bridge. By default, the reset sequence causes all devices to belong to the master host bridge. In the event of a node

or link failure, the reset sequence will cause the devices on each side of the break in the chain to belong to the host bridge on that end. Software can also reconfigure the devices to divide them more evenly between the bridges in order to balance traffic.

Because devices accept requests from both directions, they must keep track of which link incoming request packets were received on, and send any response back on the same link. An interior node may see the same SrcTag active from the host bridges at both ends of the link. The node must recognize the two host bridges as having disjoint SrcTag spaces.

Intelligent devices may also potentially be built that can override their default upstream direction and send requests in both directions. The reasons that a device would do this and its method for determining which direction to send requests are beyond the scope of this specification. If a device sends packets in different directions in the fabric, the fabric cannot make any guarantees about ordering between them.

4.1.1 LDT PWROK and RESET#

This section describes LDT PWROK and RESET# in the context of various system topologies. Section 4.1.1.1 describes requirements which all LDT devices and systems must meet. Section 4.1.1.2 describes some ~~preferred (but optional)~~host implementations.

4.1.1.1 Requirements

There must be only one LDT PWROK/RESET# signal pair for each LDT chain. These signals are inputs to each device on the chain and may also be driven by one or more devices on the chain. These signals control the powerup and reset sequence for each link interface in the chain, and may optionally control the powerup and reset sequence for other logic inside any device along the chain.

An LDT-LDT bridge device must have dedicated PWROK/RESET# pin pairs for its primary chain and for each secondary chain. The bridge must be able to drive PWROK/RESET# on its secondary chain. In addition, the bridge must pass the assertion and deassertion of PWROK/RESET# from its primary chain to its secondary chain.

4.1.1.2 ~~Preferred i~~Host Implementations

In the ~~simplest~~ case of a host with a single LDT chain, the ~~preferred implementation has the~~ host's reset signal ~~being~~ can be independent of the LDT link's PWROK/RESET# signals. This allows software running on the host to reset the LDT chain without requiring the host to be reset (see section 7.4.8.6). ~~-~~In such an implementation, the host bridge must be able to both drive and sample LDT PWROK and RESET#. In addition, the host bridge must pass the assertion of host reset to LDT RESET#. Other implementations are possible – for example host reset and LDT reset functions may be tied to a single host reset pin.

In the case of a host with multiple LDT host bridges, ~~the preferred implementation has~~ there can be independent LDT PWROK/RESET# signal pairs for each LDT chain connected to the host. As in the previous case, each of these PWROK/RESET# signals ~~is~~ can be independent of host reset. Other implementations are possible – for example host reset and all LDT reset functions may be tied to a single host reset pin.

Devices used in PC platforms have specific mandatory PWROK and RESET# requirements, described in the *LDT System Design Guide for x86 Platforms.*

4.2 Transactions and UnitID

Since all LDT transactions consist of a series of packet transfers between a device and the host bridge, the use of the UnitID field can be simply summarized with the following table

	Upstream	**Downstream**
Request	UnitID is source of request: Device's UnitID	UnitID is source of request: Host's UnitID: always zero

Response	UnitID is target of request: Device's UnitID. Bridge bit clear.	UnitID is source of request: Device's UnitID. Bridge bit set.

Peer to peer communication is implemented as a pair of LDT transactions—a transaction generated by the source device and targeted at the host, and a transaction generated by the host and directed to the target device. The UnitIDs in the request and response packets associated with these two transactions follow the rules in the table above.

4.3 Link Synchronization

Bit Time	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	11		Cmd[5:0]: 111111					
1	11111111							
2	11111111							
3	11111111							

The Sync pattern is used to indicate that a resynchronization event has occurred in the system, such as a reset or a fabric error, which requires all links to be resynchronized.

CRC checking on a link is shut down when a Sync packet is received. See section 10.1 for a description of CRC.

All fields in a Sync pattern (including the command) are all 1 bits. Transmitters on 8-bit, 16-bit, and 32-bit links generate a sync pattern by placing at least 16 bit times of all 1 bits on byte lane zero of the link. This guarantees that when a sync pattern is transmitted, the destination will recognize the pattern via its normal command decode logic even if the bit time counters on both sides have gotten out of step. Sync patterns on 4-bit and 2-bit links require two times and four times the number of all-1-bit times, respectively, as 8-bit, 16-bit, and 32-bit links.

Note that once a transmitter places a Sync pattern onto an active link, it keeps that pattern on the link until after the link is reset and synchronized. The minimum Sync-pattern times stated in the above paragraph are meant to illustrate the concept.

4.4 Requests

4.4.1 Sized Reads and Writes

Bit Time	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Mask/Count[1:0]		Compat	SrcTag[4:0]/Rsv				
3	Addr[7:2]						Mask/Count[3:2]	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

Sources use the sized read and write requests (byte or doubleword) to make requests to either memory or I/O. The data returned for sized reads cannot be coherently cached, as LDT I/O provides no coherence primitives. Sized requests contain the starting doubleword address of the data and a set of data elements to

be transferred. Bit 2 of the command indicates whether the data elements to be transferred are bytes or doublewords.

Doubleword operations can transfer any number of contiguous complete doublewords within a 64-byte aligned block. The Count field encodes the number of doubleword data elements that should be transferred, beginning at the specified address, and going in ascending order. Count codes of 0 through 15 represent 1 through 16 data elements to be transferred, respectively. Requests that cross a 64-byte boundary must be broken into multiple transactions on LDT, issued in ascending address order.

Byte reads can transfer any combination of bytes within an aligned doubleword. The Mask field is used to indicate which bytes within the doubleword are being read. Mask[0] corresponds to the lowest addressed byte, and Mask[3] corresponds to the highest addressed byte. If it is desired to issue byte-maskable reads that cross an aligned doubleword boundary, they must be broken into multiple requests, each within a single doubleword. The mask bits can be ignored for reads to regions where reads are guaranteed not to have side effects. A read for which all mask bits are zero still causes host coherence action (if to memory, and cmd[0] asserted) and still returns a RdResponse packet with one doubleword of (invalid) data.

Byte writes can transfer any combination of bytes within a naturally aligned 32-byte address region. Transfers that cross an aligned 32-byte boundary must be broken into multiple LDT transactions, issued in ascending address order. Address bits [4:2] identify the first double word of data sent in the data packet within the 32-byte region defined by address bits [39:5]. The data packet for a byte write operation contains byte mask information in the first doubleword of the data packet. The Count field is used to indicate the total size of the data packet in doublewords, including the byte masks, so it will range from one to eight to indicate two through nine doublewords to be transferred. In general it is illegal for a byte write packet to contain byte masks and no data, meaning the Count field must contain a nonzero value. The exceptions to this are interrupt and system management messages - they take the form of byte writes to predefined address regions and do not require data to be transferred. See section 3.2.2 for the format of the data packet. The count field specifies the length of the data packet independent of the value of the byte masks. Nonzero byte masks for doublewords which are not sent result in undefined behavior. Byte masks may be zero for doublewords which are sent. The entire byte mask doubleword may be zero, in which case the system performs all activities usually associated with the request. However, no data is written. If any byte masks are nonzero, the byte masks associated with the first doubleword of data must be nonzero.

The sized command field contains a bit that indicates whether or not the access requires coherence action to be taken by the system for host memory accesses. If set, the host must take whatever action is appropriate to guarantee that any caching agent remains coherent with system memory. Writes must cause caches to be updated or invalidated. Reads must return the latest modified copy of the data, even if main memory is stale. If the bit is clear, reads and writes can happen directly to and from main memory, without polling or modifying cache states.

Transactions also have an Isochronous bit associated with them that must be maintained by tunnels even when Isoc mode is disabled. Host Bridges should maintain the bit when forwarding peer-to-peer traffic if possible. See section D for details of how this is used.

Sized writes have a Posted bit. Besides serving as a virtual channel identifier, a set Posted bit indicates that the write request will receive no response in the fabric. The requester's buffer can be deallocated as soon as the write is transmitted. As such, the SrcTag field is reserved for posted requests. No assumptions can be made about the uniqueness of SrcTags for posted requests, either relative to other posted requests or other traffic.

Reads have a *response may pass posted writes* bit in the command field. This is carried with the request, but doesn't serve any purpose until the response is generated. At that time, it becomes the PassPW bit in the response. Writes do not carry the *response may pass posted writes* bit; hosts and slaves which generate a write response should generally set the PassPW bit in the response, allowing the response to pass posted requests. This is not strictly required, however.

The Compat bit is used to implement the subtractive decode necessary for legacy Southbridges. When set, it indicates that address decode in the host has found no mapping for the given access, and therefore it should be routed to the bus segment containing the Southbridge. As part of the initialization sequence, all LDT devices determine whether or not they own (or are) the Southbridge. Accesses with the Compat bit set are always accepted by devices that own the Southbridge, and ignored by all other devices, regardless of address.

4.4.2 Broadcast Message

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Reserved							
3	Addr[7:2]						Rsv	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

Broadcast messages are used by the host to communicate information to all LDT devices. They can only be issued by the host bridge, and they travel in the downstream direction the entire length of the chain, being both accepted and forwarded by all devices. Features that are implemented using broadcast messages have reserved address ranges associated with them that are recognized by all devices. All information (including potential write data) necessary to the specific type of operation being performed is carried in the address field.

Broadcasts travel in the posted channel, and the SrcTag field is reserved. No assumptions can be made about the uniqueness of SrcTags, either relative to other broadcast messages or other traffic.

4.4.3 Flush

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Rsv			SrcTag[4:0]				
3	Rsv							

Flush is designed to make sure that posted writes have been observed at host memory. It applies only to requests in the same I/O stream as the flush. It functions very similarly to a read operation, except that it returns no data. Like reads, it goes in the non-posted request virtual channel. For a flush to perform its intended function, the PassPW bit must be clear, so that the flush pushes all requests in the posted channel ahead of it. It is expected that flushes will never be issued as part of an ordered sequence, so their SeqId will always be zero. Flush requests with PassPW set or with a nonzero SeqId are legal, but their effect is unpredictable.

Note that flush only guarantees that posted requests have been flushed to their destination within the host. If the requests were peer-to-peer, this only means that they reached their destination host bridge, not the final device.

The flush response is returned from the host bridge when the requests have become globally visible in the host. Since there is no data, a TgtDone response is used.

Flush is only issued from a device to a host bridge or from one host bridge to another. Devices are never the target of a flush so they do not need to perform the intended function. If a device at the end of the chain

receives a flush, it must decode it properly to maintain proper operation of the flow control buffers and should return a TgtDone with the Error and NXA bits set.

4.4.4 Fence

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Rsv							
3	Rsv							

Fence is designed to provide a barrier between posted writes which applies across all UnitIDs and therefore all I/O streams and all virtual channels. It goes in the posted request virtual channel and has no response. There is therefore no SrcTag field in the request packet. A fence with PassPW clear will not pass anything in the posted channel regardless of UnitId. Packets with their PassPW bit clear will not pass a fence regardless of UnitId. Packets with their PassPW bit set may pass a fence. Note that while a nonposted request with PassPW clear will not pass a fence as it is forwarded through the chain, it may do so after it reaches a host bridge. See section 6.2.

For a fence to perform its intended function, the PassPW bit must be clear so that the fence pushes all requests in the posted channel ahead of it. It is expected that fence requests will never be issued as part of an ordered sequence, so their SeqId will always be zero. Fence requests with PassPW set or with a nonzero SeqId are legal, but their effect is unpredictable.

Fence is only issued from a device to a host bridge or from one host bridge to another. Devices are never the target of a fence so they do not need to perform the intended function. If a device at the end of the chain receives a fence, it must decode it properly to maintatin proper operation of the flow control buffers and should drop it. The node can choose to log this error. The error reporting method is outside the scope of this specification, however interrupt packets or sideband signaling can be used.

4.4.5 Atomic Read-Modify-Write

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Count[1:0]		Compat	SrcTag[4:0]				
3	Addr[7:3]					Rsv	Count[3:2]	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

The Atomic Read-Modify-Write (RMW) request supports two forms of atomic RMW operation on a naturally aligned quadword location.

The Fetch & Add operation is shown below.

```
FetchAdd(Out, Addr, In) {
        Out = Mem[Addr];
        Mem[Addr] = Mem[Addr] + In;
}
```

The Compare & Swap operation is:

```
CompareSwap(Out, Addr, Compare, In) {
        Out = Mem[Addr];
        If  (Mem[Addr] == Compare) Mem[Addr] = In;
}
```

The above operations must be atomically performed by the target of the request, meaning that no other agent in the system may access the addressed location between the time that it is read and written on behalf of the atomic request.

A Fetch & Add request must be accompanied by one quadword of data (the input value) and have a Count field value of one. A Compare & Swap request must be accompanied by two quadwords of data (the compare and input values) and have a Count field value of three. The compare value is first, followed by the input value. The value of the count field is used to distinguish between the two request types.

From an LDT transaction perspective, an Atomic RMW request is a nonposted write that generates a read response. The read response packet contains a single quadword – that being the original value at the addressed location. Note that for Compare & Swap, the value of the count field in the response packet is different than that in the associated request packet.

It is expected that Atomic RMW requests will be generated by LDT I/O devices or bridges and directed to system memory (DRAM) that is controlled by the host. No target other than the host is required to support atomic operations, and hosts are not required to support atomic operations to address ranges outside of system memory. If a target receives an unsupported atomic operation, it may either return a one quadword read response, with the error bit set, or it may perform the RMW in a non-atomic way.

Unlike the RdSized request packet, the Atomic RMW request packet does not contain the RespPassPW, Isoc or Coherent bits in the command field of the packet. The implied values of these bits is as follows:

- Coherent: 1. The addressed data may be cached.
- Isoc: 0. Isochronous AtomicRMW requests are not supported.
- RespPassPW: 0. The response to the AtomicRMW request may not pass posted writes.

4.5 Responses (RdResponse and TgtDone)

4.5.1 RdResponse

A node that is the target of a request for data (such as Sized Read or Read-Modify-Write) returns a read response packet to the source followed by a data packet which contains the requested data. The format of the read response packet is shown below:

Bit Time	7	6	5	4	3	2	1	0
0	Isoc	Rsv	Cmd[5:0]					
1	PassPW	Bridge	Rsv	UnitID[4:0]				
2	Count[1:0]		Error	SrcTag[4:0]				
3	Rsv		NXA	Rsv			Count[3:2]	

The Count field encodes the size minus 1 (in doublewords) of the data packet, so that intermediate nodes forwarding the response know how much data to expect. For doubleword read requests, the Count is just taken from the request packet. For byte read requests, the data field always fits within a single doubleword, so the Count field is always 0 (one doubleword). For Read-Modify-Write requests, the Count field is always 1 (one quadword).

The Error bit is used to indicate that an error occurred during the read. This could be due to the address accessed being non-existent, an ECC/Parity error in DRAM or a cache, or other problems. The requested

number of data elements are always driven to the bus, whether they are valid or not, but the Error bit indicates that the data cannot be used. A data packet with ones in all data bit positions must follow a read response packet with the Error and NXA bits set.

The Isoc bit is set to indicate that this response should flow in the ISOC response channel. See section D for details. The Isoc bit is required to be maintained even when passing through a tunnel with Isoc mode disabled. Host Bridges should maintain the bit when forwarding peer-to-peer traffic if possible.

4.5.2 TgtDone

Bit Time	7	6	5	4	3	2	1	0
0	Isoc	Rsv	Cmd[5:0]					
1	PassPW	Bridge	Rsv	UnitID[4:0]				
2	Rsv		Error	SrcTag[4:0]				
3	Rsv		NXA	Rsv				

TgtDone signals that a transaction not requiring returned data (such as Sized Write or Flush) has completed at its target. The target can release its command buffer as soon as it issues TgtDone. A non-posted request will result in either a RdResponse or a TgtDone, but not both. TgtDone also has the Error bit, similar to the one in RdResponse, which is used to indicate an error at the target in those cases where the error is detected prior to sending the response. Note that unlike read requests, write requests do not contain a PassPW bit in the command field of the request packet. Therefore the PassPW bit in the TgtDone packet will generally be set. However this is not strictly required—responders can choose to clear the PassPW bit in the TgtDone packet based on implementation-specific considerations. See appendix F.2.4 for one example.

The Isoc bit is set to indicate that this response should flow in the ISOC response channel. See section D for details. The Isoc bit is required to be maintained even when passing through a tunnel with Isoc mode disabled. Host Bridges should maintain the bit when forwarding peer-to-peer traffic if possible.

4.6 I/O Streams

LDT has the concept of I/O streams, which are groupings of traffic that can be treated independently by the fabric.

Because no peer-to-peer communication exists within the fabric, and all packets travel either to or from the host bridge, the traffic to or from each node in the fabric could in theory be treated independently by the fabric, leaving the host bridge to manage interactions between streams.

Upstream requests contain the ID of the source node, and upstream responses contain the ID of the node that generated the response. Therefore UnitID may be used to identify I/O streams for upstream packets.

Downstream responses contain the ID of the node to which the response is being sent, however downstream requests contain the ID of zero (the encoding reserved for the host bridge), not the ID of the node that is targeted by the request. Therefore it is impossible to determine independent I/O streams in downstream request traffic and it must be assumed that all downstream traffic (both requests and responses) is in the same stream.

The host bridge is responsible for managing interactions between streams. No stream information is propagated through the host bridge. The host bridge is responsible for maintaining ordering within the host domain in whatever fashion is appropriate.

It is permitted for a single physical node to be allocated multiple UnitID's if it generates multiple independent streams of traffic. This allows more concurrency among the traffic to and from that device. If this is not done, all traffic to and from that device will be ordered as a single stream, and knowledge of the possible concurrency will be lost.

4.7 Virtual Channels

LDT supports three virtual channels of information:
1. Posted Requests
2. Non-Posted Requests (reads, flushes, non-posted writes)
3. Responses

Requests may cause responses to be issued by receiving nodes. Requests received by a host bridge may also cause downstream requests to be issued (peer-to-peer reflection). Non-host bridge nodes may not make accepting a request or issuing a response due to an incoming request dependent on the ability for that node to issue an outgoing request, or receipt of a response due to a request issued by that node.

All devices must guarantee that the three virtual channels are not capable of blocking each other due to buffer management and routing issues, which is why each channel has command and data buffer space separate from the other two. However, in order to properly maintain I/O ordering, some rules are added which create dependencies between packets (in the same I/O stream) in different virtual channels. See section 6.

Note that if the chain is not partitioned between the two host bridges, there is a deadlock possibility in the double-hosted link case. A deadlocking loop can be formed if peer-to-peer requests are issued in opposite directions by two different intermediate nodes. Each reflected peer-to-peer request coming out of a host bridge can be blocked behind a stack of requests targeting the other host bridge. The host bridge will only be able to queue a finite number of peer-to-peer requests in from the link without issuing one.

LDT includes support for an optional operating mode in which the number of virtual channels is doubled to support isochronous operation. See section D for more information.

4.8 Flow Control

LDT receivers contain the following types of buffers:

1. Non-Posted Requests
2. Posted Requests
3. Responses
4. Non-Posted Request Data
5. Posted Request Data
6. Response Data

Request and Response buffers contain enough storage to store the largest control packet of that type. All data buffers can hold 64 bytes.

The table in section 3.2.1.4 defines the virtual channels and hence the buffers used for each of the control packets.

These buffers are flow controlled at the link level using a coupon-based scheme in which the transmitter contains a counter for each type of buffer at the receiver. At system reset the transmitter clears its counters, and when reset deasserts, the receiver sends Nop packets to indicate how many buffers of each type it has available. When the transmitter sends a non-info packet, it decrements the associated counter, and when a particular counter contains a zero, the transmitter stops sending packets to the associated buffer. When the receiver frees a buffer, it sends a Nop packet to the transmitter, and the transmitter increments the associated counter.

A transmitter cannot issue a control packet that has an associated data packet unless the receiver has both the appropriate control and data buffers available. If this rule is violated one virtual channel can block another and lead to deadlock, because commands with associated data packets cannot be interleaved on the link.

LDT supports an optional operation mode in which the number of virtual channels and associated flow control buffer types is doubled in order to support high priority isochronous communication. See section D for details.

It is the responsibility of nodes generating requests to be able to sink the resulting responses without other dependencies. The common way to do this is to pre-allocate enough space for all responses (including response data). Otherwise, the response and/or response data flow control buffers may get filled with responses that are not yet ready to be accepted by the internal node logic. Due to peer-to-peer requests, host bridges are exempt from this rule.

It is also required for deadlock avoidance that devices always be able to sink posted requests without any other dependencies (such as issuing cycles back to the same LDT chain or receiving responses from the chain). Due to peer-to-peer requests, host bridges are exempt from this rule.

The format of the Nop packet is shown below. Bit 7 of bit time 2 within the NOP packet is used to allow link interface hardware to differentiate an LDT I/O device from a host device that implements a superset of the LDT I/O protocol. Such a protocol could be used for the purpose of communication between devices inside the host. The link transmitter of an LDT I/O device must always place a zero in this bit position. The link receiver of an LDT I/O device may ignore it completely.

Bit Time	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Isoc	DisCon	Cmd[5:0]					
1	ResponseData[1:0]		Response[1:0]		PostData[1:0]		PostCmd[1:0]	
2	0	Diag	Rsv		NonPostData[1:0]		NonPostCmd[1:0]	
3	Rsv							

Each two-bit field in the packet indicates how many buffers of each type have become available. Hence each two-bit field can free zero, one, two, or three buffers. Receivers are not limited to having three buffers of a particular type, and they can free up additional buffers by sending additional Nop packets.

While the goal is to size each buffer at the receiver to bury the round-trip latency from the transmitted packet to the returning Nop packet, this is not strictly required by this specification. It is the responsibility of each device to guarantee that Nop packets cannot be prevented from being issued due to transmission of other traffic, to avoid starvation of the far transmitter.

If a transmitter receives more increments than it can keep track of, it must not allow its counter to wrap, but must discard the extras. This has the effect that the link will use the maximum number of buffers that both the transmitter and receiver can support. All transmitter counters must be a minimum of four bits wide, allowing up to 15 buffers to be tracked without loss.

The Diag bit is used to indicate the beginning of a CRC testing phase, as described in appendix G. Everything following the Nop packet, until the conclusion of the current CRC test interval, is ignored. This test feature is optional – receivers are not required to implement support for this test mode. Support for this mode is indicated in bit 2 of the feature capability register, described in section 7.5.9.3.

The DisCon bit is set to indicate that the link transmitter is beginning an LDTSTOP# disconnect sequence. When this bit is set all the buffer-release fields in the packet must be zero. See section 8.5 for details. Support for this bit is optional, since the LDTSTOP# feature is optional.

The Isoc bit is set to indicate that the flow control information in the associated packet pertains to the isochronous virtual channels. Isoc flow control information should only be sent and utilized when the link has Isoc Mode enabled, as described in section 7.5.4.9.

4.9 Routing

LDT has both directed and broadcast requests. Directed requests may travel in either the posted or nonposted channel; broadcast requests travel only in the posted channel. Directed packets are relayed down the fabric until they reach their destination, where they are absorbed. Broadcast packets are relayed down the entire length of the fabric, but they are also accepted at each node they pass through, and they are terminated by the node at the far end of the fabric. Broadcast packets can only be initiated by a host bridge.

4.9.1 Acceptance

A node will accept an incoming packet if any of the following are true:

1. The packet is a broadcast request.
2. The packet is a directed request with a UnitID of 0 (indicating it is from a host bridge) and (for packets with a Compat bit) the Compat bit clear, to an address owned by this node.
3. The packet is a directed request with a UnitID of 0 and a set Compat bit, and this node either is the Southbridge, or a bridge to the Southbridge.
4. The packet is a response with the Bridge bit set (indicating it is from a host bridge) and a UnitID owned by this node.

Tunnels must be able to accept downstream packets from either link in a double-hosted chain.

4.9.2 Forwarding

Whenever a node forwards a packet, it always sends it along in the direction it was travelling.

A node will forward an incoming packet to its outgoing link if any of the following are true:

1. The packet is a broadcast request.
2. The packet is a directed request with a UnitID of 0 (indicating it is from a host bridge) and (for packets with a Compat bit) the Compat bit clear, to an address not owned by this node.
3. The packet is a directed request with a UnitID of 0 and a set Compat bit, and this node is neither the Southbridge nor a bridge to the Southbridge.
4. The packet is a directed request with a non-zero UnitID (indicating that it is from an interior node).
5. The packet is a response with the Bridge bit set (indicating it is from a host bridge) and a UnitID that doesn't match this node.
6. The packet is a response with the Bridge bit clear (indicating it is from an interior node).

If the device is at the end of the fabric, that fact is indicated by the EndOfChain Configuration Space Register (CSR) bit. In that case, it will be unable to forward packets. If a packet is received that would normally be forwarded, one of the following actions is taken instead, depending on the type of packet received:

1. Broadcast requests are silently dropped—they have successfully traversed the whole fabric.
2. Non-posted downstream directed requests (those with a UnitID of zero) are responded to with a TgtDone (for Writes) or Read Response (for Reads) packet with the Error bit set, the NXA bit set, the Bridge bit clear, and a UnitId of 0. Read responses return the requested number of doublewords, with a data value of all 1 bits.
3. Non-posted upstream directed requests (those with a nonzero UnitID) are responded to with a TgtDone (for Writes) or Read Response (for Reads) packet with the Error bit set, the NXA bit set, the Bridge bit set, and a UnitId matching that of the request. Read responses return the requested number of doublewords, with a data value of all 1 bits.
4. Response and posted request packets are dropped. The node can choose to log this error. The error reporting method is outside the scope of this specification, however interrupt packets or sideband signaling can be used.

An LDT device may receive a request from one link that should be forwarded to the other link while its EndOfChain and Initialization Complete CSR bits are still clear. In this case, the device must pend the request until the Initialization Complete or LinkFailure CSR bits becomes set, indicating that the initialization attempt has completed. See section 12.3 for an example of an initialization sequence that makes use of this requirement.

See sections 7.5.4.5 and 7.5.4.6 for the definitions of the EndOfChain and Initialization Complete CSR bits and more details on how they can affect forwarding.

4.9.3 Host Bridges

Host bridges are always at the ends of the fabric, and therefore never forward packets. However, the acceptance of a packet by a host bridge will likely result in action within the host.

Host bridges take the following action upon receiving a packet:

1. Directed requests with a UnitID of 0 must be coming from another host bridge on the far end of a double-hosted chain. Type 0 configuration accesses to the device number ~~0~~ specified in the Device ID register (see section 7.5.3.3.3) are directed to the bridge's CSRs. Optionally, the host bridge could also implement a memory or I/O space region addressable from the far host bridge, to be used for messaging in clustered systems. A description of how this would be used and what it would look like is beyond the scope of this specification. In that case, the bridge would respond to accesses to this area as if it were an interior node. The responses would have the Bridge bit clear and a UnitID of 0. All requests to addresses not included above are considered accesses to nonexistent addresses. If non-posted, they are responded to with a Target Done (for Writes) or Read Response (for Reads) packet, with the Error bit set, the NXA bit set, the Bridge bit clear, and a UnitID of 0. Read responses return the requested number of doublewords, with a data value of all 1 bits. Posted requests to nonexistent addresses generate no response, and are silently dropped. This can be reported as an error via a sideband mechanism.
2. Broadcast requests must be coming from another host bridge on the far end of a double-hosted chain. They have successfully traversed the whole fabric, and may be silently dropped. Optionally, the host bridge could also implement a region addressable by broadcasts from the far host bridge. A description of how this would be used and what it would look like is beyond the scope of this specification. In that case, the bridge would handle accesses to this area as if it were an interior node, and route the broadcast to the appropriate internal target.
3. Directed requests with a nonzero UnitID are from interior nodes, and they are accepted and handled by the node logic.
4. Responses with the bridge bit set are silently dropped. This means that a host bridge tried to respond to an interior node which did not pick up the response. The node can choose to log this error and report it via a sideband mechanism.
5. Response packets with the bridge bit clear are responses to requests issued by this bridge. The host bridge will match this to one of its outstanding requests. If no match exists the node can choose to log this error and report it via a sideband mechanism.

4.9.4 Fairness and Forward Progress

In order to issue packets, a node must insert them into the stream of traffic that it is forwarding. A node must guarantee that forward progress is always made by not allowing forwarded and injected traffic to starve one another. In addition, nodes are strongly encouraged to implement a method of assuring fair access to the fabric for all nodes, approximating the round-robin behavior of a fair bus. The preferred method is described in appendix A. Some LDT devices may be used in applications where the fairness consideration is not relevant. One such example is a simple Southbridge that is always placed at the end of an LDT chain.

5 Interrupts

5.1 Interrupt Requests

All interrupt requests, regardless of interrupt class, are sent from the interrupting device to the host bridge using posted byte WrSized packets to the reserved range defined in section 9. The count field is always 0, which indicates that only a single doubleword data packet follows the write. The doubleword data packet is not used to carry byte masks; it is used to carry interrupt destination information, as described below. The table below shows the format of interrupt-request packets.

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]			Cmd[5:0]				
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Count[1:0]		Rsv	SrcTag[4:0]				
3	Rsv	DM	TM	MT[2:0]			Count[3:2]	
4	IntrDest[7:0]							
5	Vector[7:0]							
6	Addr[31:24]							
7	Addr[39:32]							
8	IntrDest[15:8]							
9	IntrDest[23:16]							
10	IntrDest[31:24]							
11	Reserved							

The host bridge is then responsible for delivery to the correct internal target or targets. These targets are identified by the value placed in the IntrDest[31:0] field.

Interrupt request packets push posted writes with the same UnitID as the interrupt request if the PassPW bit is clear. All preceding posted writes with the same UnitID as the interrupt request will be visible at their targets within the host before the interrupt is delivered.

There are three classes of interrupts supported in LDT:

- Arbitrated (Low Priority)
- Fixed
- Non-vectored

Arbitrated interrupts are only delivered to one of the addressed destinations within the host targeted by the interrupt. The ultimate target is either the lowest priority destination or a destination that is already servicing the same interrupt source (the focus processor). Arbitrated interrupts have 256 possible sources. Each interrupt source is identified by an 8-bit vector ID.

Fixed interrupts are delivered to all destinations addressed by the interrupt message. They can be used to send single, multicast, or broadcast interrupts. Fixed interrupts also have 256 possible sources, identified by vector ID.
Non-vectored interrupts carry no target information. They are always delivered to all addressed destinations. They consist of the following types:

- SMI
- NMI
- INIT
- Ext Int (Legacy PIC)

The type of interrupt is identified by the message type (MT) field, encoded as follows:

MT[2:0]	Message Type
000	Fixed
001	Arbitrated
010	SMI
011	NMI
100	INIT
101	*Startup*
110	Ext Int
111	*Reserved(EOI)*

Arbitrated and fixed interrupts can be edge triggered or level sensitive, as identified by the trigger mode (TM) field. Edge-sensitive and level-sensitive interrupts cannot be mapped to the same vector. Level-sensitive interrupts require an End Of Interrupt (EOI) message to be transmitted to acknowledge the servicing of the interrupt. A subsequent level-sensitive interrupt using the same vector will not be sent until an EOI message has been received. Edge-triggered interrupts do not signal the servicing of the interrupt. Non-vectored interrupts are always edge-triggered and therefore no LDT EOI is used.

TM is encoded as follows:

TM	Trigger Mode
0	Edge
1	Level

For all three classes of interrupts, the set of potential destinations is determined by the IntrDest and destination mode (DM) fields. The DM field determines if IntrDest represents a physical identifier or a logical identifier.

DM	Destination Mode
0	Physical
1	Logical

In physical mode IntrDest[31:8] must be zero, and each interrupt destination (processor) within the host is assigned a unique 8-bit physical ID. The physical ID 0xFF is reserved and is used to indicate that the interrupt should be broadcast to all possible destinations. A destination is considered a target for a physical mode interrupt if its ID matches IntrDest[7:0] or if IntrDest[7:0] equals 0xFF. Non-vectored interrupts always use physical mode with a IntrDest of 0xFF.

For logical addressing each interrupt destination is assigned a 32-bit logical ID. The determination of what constitutes a valid logical ID is system-specific, and the method of comparison of logical ID to IntrDest[31:0] is programmable. As an example, a system can choose a one-hot address representation, assigning one bit to each processor (limited to 32 processors), or it can define a portion of the logical address to be fully decoded and the rest of the bits to be one-hot encoded.

Startup messages are similar to Fixed interrupts in that they carry a Vector, but they have their own Message Type, TM=0, DM=0, and IntrDest=0xFF.

LDT I/O host bridges never combine multiple interrupt transactions into a single transaction within the host.

5.2 EOI

EOI messages are sent in posted broadcast message packets to all nodes across the LDT I/O fabric. Each device is responsible both for accepting the EOI and clearing outstanding interrupts to the specified vector ID, and for passing the EOI down the fabric. The format of an EOI packet is shown below:

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]			Cmd[5:0]				
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Reserved							
3	Rsv			MT[2:0]			Rsv	
4	Reserved							
5	Vector[7:0]							
6	Addr[31:24]							
7	Addr[39:32]							

EOIs use the same reserved address range as interrupt requests. The MT field must always contain the value 111 (EOI). The vector field contains the interrupt vector of the interrupt to be acknowledged.

5.3 Interrupt Acknowledge

An interrupt acknowledge transaction can be directed to the interrupt controller by performing a byte read within the reserved IACK range defined in section 9. Any read within this address range generates a RdSized request with the compat bit set. This request packet is routed directly to the Southbridge if the Southbridge is a native LDT device. If the Southbridge is implemented as a PCI device then the request packet is routed to the intervening LDT-PCI bridge. The bridge generates an interrupt acknowledge cycle on the PCI. In both cases, the read response contains the interrupt vector which will be in the least-significant byte lane position, independent of the byte masks in the RdSized request. Some systems may not require use of the Interrupt Acknowledge cycle – it is intended to provide "Programmable Interrupt Controller" (PIC) compatibility in PC systems. Nonetheless, all LDT devices must decode and forward the interrupt acknowledge request.

6 LDT I/O Ordering

This section explains the ordering rules for all three types of I/O traffic—peer-to-peer, DMA, and PIO. Peer-to-peer traffic is traffic that has both its requester and target on the LDT I/O link. Since LDT does not support peer-to-peer traffic directly, all peer-to-peer traffic (whether to the same fabric or not) goes upstream into the host and then back downstream. For purposes of ordering, the upstream and downstream legs are considered independently.

These ordering rules only apply to the order in which operations are seen by targets at the same level of the fabric hierarchy. Consider two ordered peer-to-peer write requests issued by an LDT I/O device to two different targets on different LDT I/O fabrics. The ordering rules on the originating LDT chain guarantee that the two writes reach the host bridge in the appropriate order. The host is responsible for guaranteeing that the two writes reach their target host bridges in the correct order. However, beyond that point, the writes are in independent fabrics, and there is no guarantee about the order in which they will reach their final destination. If an I/O device requires assurance of final completion, it must have a way of polling the destination device to determine that the first write has been observed before issuing the second write, or it must use non-posted writes.

Ordered operations which return responses (reads or non-posted writes) are guaranteed to complete at the target in the correct order, but no guarantee is made about the order in which the returning responses will be received. All LDT I/O devices must be able to accept responses out of order or restrict themselves to one outstanding non-posted request. A bridge that is between LDT and an I/O protocol that requires responses to be returned in order must provide sufficient buffering to be able to reorder as many responses as it may have outstanding requests.

6.1 Upstream LDT I/O Ordering

LDT recognizes three types of traffic—posted requests, non-posted requests, and responses—each in a separate virtual channel. These three types of traffic can be distinguished by their LDT command encodings. Requests have a sequence ID (SeqId) tag. Requests in the same I/O stream and virtual channel with matching non-zero SeqId's are considered part of a strongly ordered sequence. Sequences are designed to support groups of LDT transactions generated by a single request on the source I/O bus. Requests and responses both have a *may pass posted writes* (PassPW) bit.

For definitions of I/O streams and virtual channels, see sections 4.6 and 4.7, respectively.

LDT has the following upstream ordering rules:

1. Packets from different sources are in independent I/O streams and with the exception of the Fence requests, have no ordering guarantees. Devices receiving packets in different I/O streams may reorder them freely.

2. Packets in the same I/O stream and virtual channel that are part of a sequence (having matching nonzero SeqID's) are strongly ordered, and may not pass each other. Devices receiving them must keep them strongly ordered.
3. Packets in the same I/O stream, but not in the same virtual channel or not part of the same ordered sequence, have their passing rules given by the following table:

Row pass Column?	Posted Request	Non-Posted Request	Response
Posted Request	PassPW: Yes/No !PassPW: No	Yes	Yes
Non-Posted Request	PassPW: Yes/No !PassPW: No	Yes/No	Yes/No
Response	PassPW:Yes/No* !PassPW: No	Yes	Yes/No

(*): LDT implementations are strongly encouraged to allow responses with PassPW set to pass posted requests, however they cannot rely upon this behavior system-wide to ensure deadlock-free operation. Allowing responses with PassPW set to pass posted writes creates more deterministic latency on behalf of isochronous read traffic. See appendix D for more details.

Note: in the above table PassPW relates to the packet represented by the row, not the column.

No – Indicates the subsequently issued transaction is not allowed to complete before the previous transaction to preserve ordering in the system. This implies an interaction between the otherwise independent virtual channels within LDT.

Yes – Indicates the subsequently issued transaction must be able to pass the previous transaction, or deadlock may occur. This means that the packet type given in the column cannot be permitted to block the packet type given in the row at any point in the LDT fabric or host.

Yes/No – Indicates the subsequently issued transaction may optionally be allowed to complete before the previous transaction if there is any advantage to doing so. There are no ordering requirements between the two transactions. However, support for reordering is not required—failure to reorder the packets will not lead to deadlock.

6.2 Host Ordering Requirements

The host bridge and host system are required to preserve the virtual channels provided in the LDT I/O fabric as defined in section 6.1, and to guarantee that transactions that are ordered within the LDT fabric are ordered within the host. This means that for an ordered pair of transactions, the second transaction cannot be issued by the host bridge until the first transaction has reached its ordering point. The definition of this ordering point depends on the type of transactions in the ordered pair and the relationship of their targets. However, in the case of posted peer-to-peer I/O operations, the host can only guarantee that the first operation has been issued on its destination link; it has no way of knowing whether the operation has reached its final target device.

The rules governing the host's processing of ordered LDT I/O transactions are expressed in the table below. The table defines what ordering point the first request in an ordered pair must reach before the second request can be safely started. There are three defined ordering points, from earliest to latest. (Some of these may coincide for particular transaction types, depending on the host implementation.)

- ***Locally Ordered (LO)*** -- The first transaction has reached a point where its ordering is guaranteed relative to all subsequent requests from this same source to the same destination. If the host starts another request to the same destination, the destination will get them in order.
- ***Globally Ordered (GO)*** -- The first transaction has reached a point where it is guaranteed to be observed in the correct order (relative to the second transaction) from any observer.
- ***Globally Visible (GV)*** -- The first transaction is visible to all processors. That is, any processor read will return the new data. This means that in addition to being globally ordered, all cache state transitions initiated by the first transaction have completed.

First Command ~~Request~~	Second Command ~~Request~~	First Target same as Second?	Second ~~Request~~ Command waits for the First to be:
~~Memory~~ Cacheable Wr	~~Memory~~ Cacheable Wr	--	GV
Cacheable Wr ~~Memory Wr~~	Cacheable Rd ~~Memory Rd~~	Y	LO
Cacheable Rd ~~Memory Rd~~	Cacheable Rd or Wr ~~Memory Rd or Wr~~	Y	LO
Cacheable Wr ~~Memory Wr~~	Cacheable Rd ~~Memory Rd~~	N	GO
Cacheable Rd ~~Memory Rd~~	Cacheable Rd or Wr ~~Memory Rd or Wr~~	N	GO
~~I/O~~Non-Cacheable	Non-Cacheable~~I/O~~	Y	LO
Non-Cacheable~~I/O~~	Non-Cacheable~~I/O~~	N	GO
Cacheable Wr ~~Memory Wr~~	Non-Cacheable~~I/O~~	--	GV
Cacheable Rd ~~Memory Rd~~	Non-Cacheable~~I/O~~	--	GO
Non-Cacheable~~I/O~~	Cacheable~~Memory~~	--	GO
Cacheable Wr ~~Memory Wr~~	Flush/Intr/SM/Resp	--	GV
Cacheable Rd ~~Memory Rd~~	Flush/Intr/SM/Resp	--	No wait requirements
Non-Cacheable~~I/O~~	Flush/Intr/SM/Resp	--	GO
Flush/Response	Any	--	--
Intr/SM	Response	--	GV
Intr/SM	Any but response	--	No wait requirements
Posted Cacheable ~~memory~~	Fence	--	GV
Posted Non-Cacheable ~~I/O~~	Fence	--	GO
Intr/SM	Fence	--	GV
Any nonposted	Fence	--	No wait requirements
Fence	Any posted	--	GV
Fence	Any nonposted or Resp	--	No wait requirements

~~Note: In the table above, *Intr*, includes both interrupt and system management transactions.~~

6.2.1 Host Responses to Non-Posted Requests

The host cannot generate a response to a non-posted request until all side effects of the request are globally visible. For a memory request this means that all cache state transitions initiated by the request have been completed. For I/O requests this means that data writes or read side effects have occurred. A response to a non-posted request implies that all previous ordered requests to memory are globally visible. It also implies that all previous ordered requests to I/O have been globally ordered, but it cannot be assumed that they are globally visible.

6.3 Downstream LDT I/O Ordering

The rules for downstream ordering are the same as those for upstream ordering, with the exception that I/O streams are identified by the target of the transaction, rather than the source. The same three virtual channels exist. However, UnitID may not be used to identify unique I/O request streams in the downstream direction, so it must be assumed that all downstream traffic is in the same stream. This asymmetry in the definition of I/O streams for upstream and downstream traffic is why it is important for a node to be able to differentiate upstream responses from downstream responses, and provides the motivation for the Bridge bit in the response packet.

The host must also guarantee that peer-to-peer LDT traffic that was part of an ordered sequence when received is also emitted downstream as an ordered sequence.

6.4 Ordering in Double-Hosted Chains

In general, upstream and downstream traffic moving in the same direction along an LDT chain have no ordering dependencies with respect to each other, as they are guaranteed to be in different I/O streams. The exception is the case of communication directly between host bridges at opposite ends of a double hosted chain. In this case, requests from one host bridge to the other are always travelling downstream, and responses from that host bridge are travelling upstream.

In the event that one host bridge (bridge A) issues a posted write to the other (bridge B), and bridge B issues a read request to A, the read response will be travelling in the same direction as the posted write. Despite the fact that the request is moving downstream and the response upstream, they must be treated as being in the same I/O stream (the response must push the request if passPW is clear) in order to support producer/consumer communication between the hosts.

In this case, both the request and response will contain a unitId of 0. So, this requirement can be supported simply by doing ordering checks based solely on unitId for upstream responses, and excluding information about whether the request they are checking against is moving upstream or downstream.

7 Configuration Accesses

LDT implements configuration space similarly to PCI, as defined in the *PCI Local Bus Specification, Revision 2.2*. LDT devices and bridges (including host bridges) must implement appropriate PCI configuration headers. Buses and devices are numbered in a fashion that maps into PCI bus and device numbers. Configuration software should be able to do configuration across LDT in a way that is indistinguishable from an equivalent PCI bus hierarchy. Configuration space is mapped to a predefined region of the LDT system address space. See section 9 for details.

7.1 Configuration Cycle Types

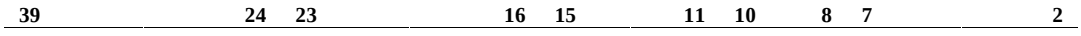
PCI uses two types of configuration cycles, type 0 and type 1. The two types are needed because it must be possible to access the configuration space of devices on a bus without the devices knowing what bus they are on.

Type 0 cycles are used to access devices on the current bus. They contain a function number and register number. The bus number is implicitly the current bus. The device number is indicated by the IDSEL# pins, which are asserted as appropriate by the bridge. Therefore, PCI devices do not need to know their bus number or device number in order to respond to configuration accesses.

Type 1 cycles are used to transmit configuration cycles over intermediate buses, and they contain bus number, device number, function number, and register number fields. Bridges forward type 1 cycles through the bus hierarchy, and translate them to type 0 cycles when driving them onto their final destination bus. Host bridges can optionally implement the capability to transmit PCI special cycles to remote buses using device 31, function 7, register 0, type 1 configuration cycles.

LDT also requires two types of configuration cycles, for the same reasons as PCI.

An LDT Type 0 access is performed by issuing a RdSized or nonposted WrSized request with an address of the following form. They are only issued by host bridges, and therefore always travel downstream. Unlike PCI, LDT Type 0 accesses contain the device number, because all LDT devices know what their device number is. LDT has no analog of the IDSEL# lines. Host bridges that support double-ended links will respond to type 0 accesses on their secondary interface at the Device Number specified in Host Interface Command register. See section 7.5.3.3.3. Note that in a double-hosted link, this implies that both bridges could be responding to the same address—which one you are talking to is determined by which direction the packets are travelling. This function is only intended to be used by system-sizing firmware. .



FDFEh	Reserved	Device Number	Function Number	Register Number

An LDT Type 1 access is performed by issuing a RdSized or nonposted WrSized request with an address of the following form. In general, type 1 accesses are issued by host bridges. The only type 1 accesses that can flow upstream from a slave device are Special Cycle requests. The system's response to all other upstream type 1 requests is undefined.

39 24	23 16	15 11	10 8	7 2
FDFFh	Bus Number	Device Number	Function Number	Register Number

RdSized and WrSized configuration accesses of greater than one doubleword are not supported. Posted configuration writes are not allowed and their effect is undefined.

7.2 Configuration Space Mapping

7.2.1 Function and Register Numbering

The numbering of functions and registers within a device is device-specific, except that every implemented device number must have a function 0 that contains a standard PCI configuration header that identifies the device. Certain other standard Configuration Space Registers (CSRs) are required by the LDT specification.

7.2.2 Device Numbering

Devices in LDT are identified by UnitIDs, which range from 00h to 1Fh. A single physical device can own multiple UnitID values. Every LDT device owns the device numbers that correspond to its UnitIDs, and it must implement a configuration space (with configuration header) at the Device Number equal to its lowest UnitID value. It may choose to implement configuration spaces corresponding to any number of its remaining UnitIDs, including none. Each implemented space must contain an appropriate PCI header. Unimplemented spaces must not be responded to by the device. Accesses to these device numbers will not be accepted by any device on the LDT chain, and so will receive a response with the error and NXA bits set.

As described in appendix E.3, some systems require a compatibility chain that is enumerated as bus 0. In such a system, any configuration space registers implemented in the bus zero space by the host must appear in the uppermost device numbers on that bus. In such a case, the number of devices (and therefore UnitIDs) available to the LDT chain implementing bus 0 is reduced accordingly. If the host were to occupy device 0, then the LDT chain could not be enumerated.

As described in appendix, F.4, some legacy operating systems may require AGP configuration registers to be implemented in the Bus 0, Device 0 range, in which case the host's configuration space registers must appear somewhere other than that range. Even though the AGP CSRs appear in Device 0, UnitID 0 is still reserved for host use so that devices can distinguish upstream cycles from downstream cycles, as described in section 4.9. This requires the AGP bridge to consume at least one extra UnitID, as it cannot use UnitID 0.

For LDT hosts that also implement AGP configuration space and require legacy OS compatibility, the host can either:

1) Make configuration space relocatable so that LDT bus enumeration may occur.
2) Place configuration space in the uppermost device numbers and provide a mechanism for hiding device 0 registers to allow LDT bus enumeration.

The means to do either of the above, or perhaps other solutions to this problem, are implementation-specific and beyond the scope of this specification.

7.2.3 Bus Numbering

Each LDT chain in the system is assigned a single bus number. For double-hosted physical chains, which are logically partitioned between two host bridges, each logical chain has a separate bus number. Bus numbers are assigned by system initialization software at reset, and follow the conventions used for PCI bus numbering, which require that the bus tree be numbered in a depth-first fashion. No distinction is made between PCI buses and LDT chains in the numbering.

7.3 LDT Device Header

Devices that sit on LDT and perform non-bridging functions implement PCI device headers, as shown below. The fact that this is a device header is contained in the header type register. Connecting to two links in an LDT chain does not constitute a bridging function. Fields in an LDT device header are the same as defined in the *PCI Local Bus Specification, Revision 2.2*, with the exceptions listed in the sections below.

31 24	23 16	15 8	7 0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency Timer	Cache Line Size	0Ch
Base Address Registers				10h 14h 18h 1Ch 20h 24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Capabilities Pointer	34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch

Some fields in the headers and LDT capability blocks are defined to be ***persistent*** through warm reset, which means their values are only affected by cold (power-on) reset. If not specified otherwise, a field is ***non-persistent***, which means that the field is initialized by warm reset.

7.3.1 Command Register: Offset 04h

The following bits are implemented in an LDT device's command register. All other bits are not applicable to LDT, and must be hardwired to 0.

7.3.1.1 I/O Space Enable (bit 0)

This bit must be set for the device to accept any non-compatibility accesses (requests with the compat bit clear) to the ~~PCI~~ I/O Address Space, as given in section 9. If this device is a subtractive decode device, requests with the Compat bit set will still be accepted, regardless of the state of this bit.

7.3.1.2 Memory Space Enable (bit 1)

This bit must be set for the device to accept any non-compatibility accesses to the Memory-Mapped I/O Address Space, as given in section 9. If this device is a subtractive decode device, requests with the Compat bit set will still be accepted, regardless of the state of this bit.

7.3.1.3 Bus Master Enable (bit 2)

This bit must be set to allow the device to issue requests onto the LDT chain. Configuration, system management, and interrupt requests are issued irrespective of the state of this bit. All other requests require

this bit to be set in order to be issued. Requests from other devices on the chain may still be forwarded, independent of the state of this bit.

7.3.1.4 SERR# Enable (bit 8)

If set, the device will flood all its outgoing links with sync packets in the event that it detects a fatal error. If clear, the device may not generate sync packets except as part of initial link synchronization, although it can still propagate them from one link to the other. This bit is read/writable by software, and resets to 0.

7.3.2 Status Register: Offset 06h

The following bits are implemented in an LDT device's status register. All other bits are not applicable to LDT, and must be hardwired to 0.

7.3.2.1 Capabilities List (bit 4)

This read-only bit will always be set to 1, to indicate that the device has a capabilities list containing (at least) LDT-specific configuration information.

7.3.2.2 Signaled Target Abort (bit 11)

This bit is set by an LDT device that returns an error response to a transaction addressed to it. This bit does not get set by an EndOfChain device when it returns an NXA error response due to a request not being accepted by any devices. This bit is persistent through warm reset. The state of this bit after cold reset is 0. It can be cleared by software by writing a 1 to it.

7.3.2.3 Received Target Abort (bit 12)

This bit is set by an LDT device that receives an error response (other than an NXA response) to a request it issued. This bit is persistent through warm reset. The state of this bit after cold reset is 0. It can be cleared by software writing a 1 to it.

7.3.2.4 Received Master Abort (bit 13)

This bit is set by an LDT device that receives an NXA error response to a request it issued. This bit is persistent through warm reset. The state of this bit after cold reset is 0. It can be cleared by software writing a 1 to it.

7.3.2.5 Signaled System Error (bit 14)

This bit is set by an LDT device that has flooded the link with sync packets to signal a system error. This bit is persistent through warm reset. The state of this bit after cold reset is 0. It can be cleared by software by writing a 1 to it. However, software will not be able to access the device via the flooded link until a reset has occurred.

7.3.3 Cache Line Size Register: Offset 0Ch

This register is not implemented by LDT devices.

7.3.4 Latency Timer Register: Offset 0Dh

This register is not implemented by LDT devices.

7.3.5 Base Address Registers (BARs): Offsets 10-24h

Base address registers for LDT devices are implemented as described in the PCI Spec. Incoming addresses in the Memory-Mapped I/O Space address range (as described in section 9) are compared directly to the values in memory space BARs. If the memory space BARs are programmed to support 64-bit addressing, then the incoming addresses from LDT must be zero extended to 64 bits before being compared to the BAR value. Incoming addresses in the ~~PCI~~ I/O Space address range (which is only a 25 bit space) have only their

bottom 25 bits compared to the I/O space BARs. Bits 31:26 of the I/O space BAR must be 0 for a match to occur.

7.3.6 CardBus CIS Pointer: Offset 28h

This register is not implemented by LDT devices.

7.3.7 Capabilities Pointer: Offset 34h

Every LDT device has a capabilities pointer to a linked capabilities list that contains (at least) the LDT-specific capability registers.

7.3.8 Interrupt Line, Interrupt Pin Registers: Offsets 3C & 3Dh

The interrupt line and interrupt pin registers have the same definition for LDT as for PCI. See the *PCI Local Bus Specificiation, Revision 2.2,* for details.

7.3.9 Min_Gnt, and Max_Lat Registers: Offsets 3E & 3Fh

These registers are not implemented by LDT devices, and they return 0's if read.

7.4 LDT Bridge Headers

Devices that bridge between LDT and other bus protocols (including subsidiary LDT chains) implement a PCI bridge header as described in the *PCI-to-PCI Bridge Architecture Specification Revision 1.1*, with the exceptions listed below. Note that LDT can be the primary or secondary bus of the device, or both. Some register bits have LDT-specific meanings only when LDT is connected to a specific port of the bridge. If that interface is to a non-LDT bus, the requirements of that bus protocol determine the meaning of the bit. Unless otherwise noted, in the case of a host bridge implementation, the registers described in this section are reset by host reset and unaffected by LDT RESET#. In the case of an LDT-LDT bridge implementation, the registers are reset by LDT RESET# on the bridge's primary chain, and are unaffected by LDT RESET# on the bridge's secondary chain.

31 24	23 16	15 8	7 0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Primary Latency Timer	Cache Line Size	0Ch
Base Address Register 0				10h
Base Address Register 1				14h
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	18h
Secondary Status		I/O Limit	I/O Base	1Ch
Memory Limit		Memory Base		20h
Prefetchable Memory Limit		Prefetchable Memory Base		24h
Prefetchable Base Upper 32 Bits				28h
Prefetchable Limit Upper 32 Bits				2Ch
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		30h
Reserved			Capabilities Pointer	34h
Expansion ROM Base Address				38h
Bridge Control		Interrupt Pin	Interrupt Line	3Ch

7.4.1 Command Register: Offset 04h

All of the command register bits implemented in the Device Header Command Register (section 7.3.1), are implemented in bridges that have LDT on their primary bus, and they affect operation only on the primary

bus. If the Bus Master Enable bit is cleared in a bridge device, preventing forwarding of transactions to the primary bus, transactions that would be forwarded must be master-aborted on the secondary bus. If the secondary bus is LDT, this is indicated by returning a response with the NXA bit set.

In addition, bridge devices have the option of implementing the VGA Palette Snoop Enable bit (bit 5). If not implemented, it is hardwired to 0. This bit functions similarly to the way it is described in *PCI-to-PCI Bridge Architecture Specification Revision 1.1*. If the bit is set, WrSized operations to addresses in the first 64 Kbytes of the ~~PCI~~ I/O address range (as defined in section 9) have address bits 9:0 compared to addresses 3C6h, 3C8h, and 3C9h. (Address bits 1:0 are not included in WrSized packets, and so must be determined from the address of the least significant enabled byte in the packet.) Accesses on a primary LDT bus that match are forwarded to the secondary bus of the bridge. Accesses on a secondary LDT bus that match are not accepted, regardless of the state of other address decode registers.

7.4.2 Status, Cache Line Size, Primary Latency Timer, Base Address, Interrupt Pin, and Interrupt Line Registers

For a bridge with LDT as its primary bus, these registers are implemented the same way as the corresponding registers in an LDT device header. They are not related to the secondary bus.

7.4.3 Secondary Latency Timer Register: Offset 1Bh

If the secondary bus is LDT, this register is reserved.

7.4.4 Secondary Status Register: Offset 1Eh

If the secondary bus is LDT, most of the bits defined in the in *PCI-to-PCI Bridge Architecture Specification Revision 1.1* are not relevant and are reserved, being hardwired to 0. The exceptions are listed below:

7.4.4.1 Signaled Target-Abort (bit 11)

If set, this bit indicates that the bridge has issued a non-NXA error response on the secondary bus. This bit is persistent through warm reset. It is cold reset to 0, and can be cleared by software by writing a 1 to it.

7.4.4.2 Received Target-Abort (bit 12)

If set, this bit indicates that the bridge has received a non-NXA error response from the secondary bus. This bit is persistent through warm reset. It is cold reset to 0, and can be cleared by software by writing a 1 to it.

7.4.4.3 Received Master-Abort (bit 13)

If set, this bit indicates that the bridge has received an NXA error response from the secondary bus. This bit is persistent through warm reset. It is cold reset to 0, and can be cleared by software writing a 1 to it.

7.4.4.4 Detected System Error (bit 14)

If set, this bit indicates that the bridge detected sync packet flooding on its secondary bus. This bit is persistent through warm reset. It is cold reset to 0, and can be cleared by software writing a 1 to it.

7.4.5 Memory and Prefetchable Memory Base and Limit Registers: Offsets 20-2Ch

For accesses coming in on an LDT bus, these registers are only compared to addresses in the Memory-Mapped I/O range, as defined in section 9. If the memory space BARs are programmed to support 64-bit addressing, then the incoming addresses from LDT must be zero extended to 64 bits before being compared to the BAR value. Matching addresses are forwarded from the primary to the secondary bus, and ignored on the secondary bus. Non-matching addresses are forwarded from the secondary to the primary bus, and ignored on the primary bus.

7.4.6 I/O Base and Limit Registers: Offsets 1C, 1D, 30, & 32h

For accesses coming in on the primary (LDT) bus, these registers are only compared to addresses in the 32-Mbyte ~~PCI~~ I/O range, as defined in section 9. Only the low 25 bits of the incoming byte address are used. All bits above bit 24 are forced to 0 before the comparison. Matching addresses are forwarded from the primary to the secondary bus and ignored on the secondary bus. Non-matching addresses are forwarded from the secondary to the primary bus and ignored on the primary bus.

7.4.7 Capabilities Pointer Register: Offset 34h

Every bridge with LDT on at least one port has a capabilities pointer to a linked capabilities list that contains (at least) the LDT-specific capability registers.

7.4.8 Bridge Control Register: Offset 3Eh

All unspecified bits are reserved, and return 0 if read.

7.4.8.1 Parity Error Response Enable (bit 0)

If the secondary bus is LDT, this bit is reserved.

7.4.8.2 SERR# Enable (bit 1)

This bit controls the mapping of system errors from the secondary to the primary bus of the bridge. If set, and the SERR# Enable in the Command register is set, system errors will propagate. System errors in LDT are indicated by flooding the chain with sync packets.

7.4.8.3 ISA Enable (bit 2)

This bit is implemented similarly to the way it is described in the *PCI-to-PCI Bridge Architecture Specification Revision 1.1*. For LDT requests in the bottom 64 Kbytes of ~~PCI~~ I/O space (see section 9), this modifies the response to accesses that hit in the range defined by the I/O Base and Limit registers.

Implementation of this bit is optional – if it is not implemented is should read as zero.

7.4.8.4 VGA Enable (bit 3)

This bit functions similarly to the way it is described in the *PCI-to-PCI Bridge Architecture Specification Revision 1.1*. If enabled, RdSized and WrSized operations within the address range 0_000A_0000h to 0_000B_FFFFh (inclusive) or within the first 64 Kbytes of the ~~PCI~~ I/O range (as defined in section 9), with address bits 9:0 in the (inclusive) range 3B0h to 3BBh or 3C0h to 3DFh, are forwarded from the primary to the secondary interface, and are ignored on the secondary interface, overriding the values in the Memory Base and Limit and I/O Base and Limit registers.

Implementation of this bit is optional – if it is not implemented is should read as zero.

7.4.8.5 Master-Abort Mode (bit 5)

This bit controls the behavior on the source bus when a transaction forwarded through the bus receives a master abort on the target bus. In LDT, master aborts are indicated by a request receiving NXA error response.

If the Master-Abort Mode bit is set, and a nonposted request forwarded from an LDT chain receives a master abort on the target bus, the LDT request will receive an error response with the NXA bit clear. (That is, it will be signaled as an internal error on the originating chain.) If the request was posted, it is not possible to return an error response. This can be signaled by a sideband method.

If the Master-Abort Mode bit is clear, the request will appear to complete normally on the originating LDT chain. Writes will receive TargetDone. Reads will receive ReadResponses, with the appropriate amount of data, which will be all hex Fs.

7.4.8.6 Secondary Bus Reset (bit 6)

This bit allows software to reset the secondary bus of the bridge. Writing a one to this bit causes the bridge to force the secondary bus into reset. Writing a zero to this bit cause the bridge to stop forcing the secondary bus into reset. The remaining details are dependent on the type of secondary bus.

This bit is required for bridges that have LDT on their primary interface. This bit is also required for host bridges in which LDT RESET# is independent of host reset.

If the secondary bus of the bridge is an LDT chain, writing a 1 to this bit will cause the RESET# line for that chain to be asserted. If the Warm Reset bit in the Host/Secondary Interface Command Register (section 7.5.3.3.1) is clear, the PWROK line for that chain will also be deasserted. When, after being initially set, the Secondary Bus Reset bit is cleared, the chain will come out of reset. If the Warm Reset bit is set, this simply results in the deassertion of RESET#. It is the responsibility of software to delay deasserting the reset long enough to satisfy the RESET# pulse width requirement. If the Warm Reset bit is clear, clearing the Secondary Bus Reset will cause PWROK to assert. Hardware will then wait for the appropriate amount of time and deassert the RESET# pin. If software wishes to be able to determine that the bus has come out of reset, it can poll the Initialization Complete bit of the Link Control Register (section 7.5.4.5).

7.4.8.7 Fast Back-to-Back Enable, Primary Discard Timer, Secondary Discard Timer, Discard Timer Status, Discard Timer SERR# Enable (bits 11:7)

All of these bits are controls for the secondary interface of the bridge. If the secondary interface is LDT, they are reserved, and hardwired to 0.

7.5 LDT Capability Registers

LDT-specific configuration and status information is mapped into configuration space using the Capabilities List methodology described in the the *PCI Local Bus Specification, Revision 2.2.* A device with multiple LDT interfaces (e.g., an LDT-LDT bridge, with LDT on both the primary and secondary interface) must implement one capability block for each interface. So, a single-link device would have a single primary interface block containing one link control register and a tunnel device would have a single primary interface block containing two link control registers. A bridge would have one primary interface block containing one or two link control registers (depending on whether or not the device provides a tunnel to allow a chain to continue on the primary bus) in addition to one secondary interface block for each LDT bridge. An LDT Bridge Header and Secondary Interface Capability Block are required for each chain. Only one Primary Interface Capability Block is required for a device that uses LDT as its primary interface. If a device is a host bridge or has a different bus for its primary interface, only secondary interface block(s) are required and Bridge Headers are optional.
The layout of the capabilities block is determined by the value in the Capability Type field in the command register, but the Capability ID register, Capabilities Pointer register, and Capability Type field are always the same. The offset that the block begins at is implementation specific.

The layout of a Slave/Primary Interface block is as follows:

<table>
<tr><th>31 24</th><th>23 16</th><th>15 8</th><th>7 0</th><th></th></tr>
<tr><td colspan="2">Command</td><td colspan="2">Capabilities Pointer</td><td>Capability ID</td><td>+00h</td></tr>
<tr><td colspan="2">Link Config 0</td><td colspan="3">Link Control 0</td><td>+04h</td></tr>
<tr><td colspan="2">Link Config 1</td><td colspan="3">Link Control 1</td><td>+08h</td></tr>
<tr><td colspan="2">LinkFreqCap0</td><td>Rsv</td><td>Link Freq 0</td><td>LDT Rev ID</td><td>+0Ch</td></tr>
<tr><td colspan="2">LinkFreqCap1</td><td>Rsv</td><td>Link Freq 1</td><td>Feature</td><td>+10h</td></tr>
</table>

The layout of a Host/Secondary Interface block is as follows:

31 24	23 16	15 8	7 0	
Command		Capabilities Pointer	Capability ID	+00h
Link Config		Link Control		+04h
LinkFreqCap		Rsv / Link Freq	LDT Rev ID	+08h
Reserved		Rsv	Rsv / Feature	+~~10h~~0Ch

Unless otherwise noted, in the case of a host bridge implementation, the registers described in this section are reset by host reset and unaffected by LDT RESET#. In the case of an LDT-LDT bridge implementation, the registers are reset by LDT RESET# on the bridge's primary chain, and are unaffected by LDT RESET# on the bridge's secondary chain.

7.5.1 Capability ID: Offset 00h

This is a read-only register. The capability ID for LDT is 08h.

7.5.2 Capabilities Pointer: Offset 01h

This read-only register contains a pointer to the next capability in the list, or a value of 00h if this is the last one.

7.5.3 Command Register: Offset 02h

Type 15 13	12	11	10	9 7	6 5	4 2	1	0
Slave/Pri	000	Rsv	Default Direction	Master Host	Unit Count		Base UnitID	
Host/Sec	001	Reserved				Device ID	Double Ended	Warm Reset

Command[15:0] contains bits used to configure the LDT interface. All unspecified bits are reserved, and are hardwired to 0.

7.5.3.1 Capability Type (bits 15:13)

This field is read-only, and indicates the type of information present in this capability block.

Currently, only two valid encodings are defined. The rest are reserved.

Encoding	Capability Type
000	Slave or Primary Interface
001	Host or Secondary Interface

Primary and Secondary Interface encodings indicate that this block is used to configure the primary (including the interface of an LDT slave device) or secondary (including the interface of a host bridge) interface of a device, respectively.

The layout of the rest of the bits of the Command Register depends on the Capability Type field, as given below.

7.5.3.2 Slave/Primary Interface Command Bits

7.5.3.2.1 Base UnitID (bits 4:0)

This register contains the lowest numbered UnitID belonging to this device. If the device owns multiple UnitIDs, the additional ones occupy the next consecutive UnitID values above the base. The contents of this field are used to generate the UnitID field in request and response packets issued by this device,

identify responses returning to this device, and identify configuration requests directed to this device. It is readable and writable from software. Its value at reset is 00h.

7.5.3.2.2 Unit Count (bits 9:5)

This read-only field contains the number of UnitIDs this device requires. Therefore, the highest UnitID used by this device is given by (BaseUnitID + UnitCount – 1). If the highest UnitID used exceeds 1Fh, the behavior of the device is undefined.

7.5.3.2.3 Master Host (bit 10)

This bit indicates which link is the path to the master (or only) host bridge on the LDT chain. It is readable from software, but not directly writable. Any time the command register is written, this bit is loaded with the link number that the write came from. Its value at reset is 0.

7.5.3.2.4 Default Direction (bit 11)

This bit determines the default direction for an LDT device to send requests. A 0 indicates requests should be sent toward the master host bridge, as indicated by the Master Host bit. A 1 indicates requests should be sent in the opposite direction. An intelligent device can choose to ignore this if it bases the direction of DMA upon some other criteria. This bit can be read and written by software. It is initialized to 0 at reset.

7.5.3.3 Host/Secondary Interface Command Bits

7.5.3.3.1 Warm Reset (bit 0)

This optional bit allows a reset sequence initiated by the Secondary Bus Reset bit of the Bridge Control Register (see section 7.4.8.6) to be either warm or cold. The contents of this bit are only looked at when a software reset sequence is initiated. If it is 0, PWROK will be driven low as part of the sequence, causing a cold reset. It is the responsibility of the hardware to sequence PWROK and RESET# correctly. If this bit is implemented, it is readable and writable by software and its value after reset is 1. If not implemented, this bit is read-only, and hardwired to 1. Changing the state of the Warm Reset bit while the Secondary Bus Reset bit is asserted results in undefined behavior.

7.5.3.3.2 Double Ended (bit 1)

This bit indicates that there is another bridge at the far end of the LDT chain. It is readable and writable by software, and resets to 0. For bridges that don't support double-ended chains, this bit must be hardwired to 0.

7.5.3.3.3 Device ID (bits 6:2)

This optional register contains the device ID of configuration accesses that the host bridge responds to in double-hosted chains. While this will typically be 0, in some cases there may be legacy software considerations that require host configuration space registers to be located somewhere other than in the Bus0, Device0 space. See section 7.2.2. If this bit is implemented, it is readable and writeable by softwares and its value after reset is 0. If not implemented this register is read-only and hardwired to 0.

7.5.4 Link Control Register: Offsets 04h & 08h

Host/secondary interface blocks implement one copy of this register; slave/primary interface blocks implement 2, one for each link. For devices that only implement one link in the chain, all bits of the second control register are reserved and hardwired to 0, except for the Link Failure, End of Chain, and Transmitter Off fields, which are hardwired to 1.

15 14	13	12	11 8	7	6	5	4	3	2	1	0
Rsv	LSEn	IsocEn	CRC Error	TXO	EOC	Init	LkFail	CFE	CST	CFlE	Rsv

7.5.4.1 CRC Flood Enable (bit 1)

If clear, this bit prevents CRC errors from generating sync packets and causing the system to come down, and from setting the LinkFail bit. However, CRC checking logic still runs on all lanes enabled by LinkWidthIn, and detected errors still set the CRC Error bits. Its reset value is 0.

7.5.4.2 CRC Start Test (bit 2)

When this optional bit is written to a 1 by software, the hardware initiates a CRC test sequence on the link, as described in appendix G. When the test sequence is complete, and CRC has been checked on all CRC intervals containing test pattern data, hardware clears the bit. Software can determine that the test has completed by reading the bit and checking the status of the CRC Error bits. Implementation of CRC test pattern generation is optional. If not implemented, this bit must be hardwired to 0. This bit resets to 0. Software should not set this bit unless it has checked the CRC Test Mode Capability bit, as defined in section 7.5.9.3, of the device on the other side of the link.

7.5.4.3 CRC Force Error (bit 3)

When this bit is set, bad CRC is generated on all transmitting lanes, as enabled by LinkWidthOut. The covered data is not affected. This bit is readable and writable from software, and resets to 0.

7.5.4.4 Link Failure (bit 4)

The LinkFail bit is set if a failure has been detected on the link. Devices with only one LDT link will hardwire LinkFail for the nonexistent link to one. Devices that contain an interface that is not used by the system will also set LinkFail for the unused link to one. As described in section 12.2, devices can identify unused links because their CAD[0] input is tied to a logical zero. This register is persistent through warm reset. If the bit is set, reset hardware does not attempt to synchronize that link; it continues to drive the link reset value, as defined in section 12.2. This prevents the Initialization Complete bit from getting set on either end of the link. Implemented LinkFail bits are reset to 0 on a cold reset.

LinkFail bits are set by hardware in the event of a CRC error (assuming CRC Flood Enable is set). The bits are also software writable, allowing software to disable links or re-enable failed links. Clearing a set LinkFail bit has no effect except to allow that link to be initialized on the next warm reset. Setting a clear LinkFail bit, once link initialization is complete, has no effect except to prevent that link from being initialized on the next warm reset.

7.5.4.5 Initialization Complete (bit 5)

This read-only bit is reset to 0, and set by hardware when low-level link initialization sequence (see section 12.2) is successfully complete in both the receiver and transmitter. If there is no device on the other end of the link, or if that device is unable to properly perform the low-level link initialization protocol, the bit never gets set. ~~CRC checking in the hardware does not begin until initialization is complete.~~ A device may receive a request before initialization of all the attached links is complete. If the request needs to be forwarded to an uninitialized link, the device must pend the request until the link is successfully initialized or LinkFail is set. It is possible that neither of these events occurs, and as a result the request could hang. Hosts that use the initialization sequence described in section 12.3.1 are encouraged to implement a timeout counter to prevent a system-wide initialization error due to link-level initialization problems on a non-default chain.

For host interface blocks, this bit is reset by LDT RESET#, not host reset. For secondary interface blocks, this bit is reset by LDT RESET# on the bridge's secondary chain, not LDT RESET# on the primary chain.

7.5.4.6 End of Chain (bit 6)

The EndOfChain bit is set to indicate that the given link is not part of the logical LDT chain. Packets which are issued or forwarded to this link are either dropped or result in an NXA error response, as appropriate (see section 4.9.2). Packets received from this link are ignored, and CRC is not checked. If the transmitter is still enabled (TransmitterOff CSR bit is clear) when the EndOfChain bit is set, the transmitter

must drive NOP packets (all CAD bits 0, with CTL asserted) with good CRC. This is required to prevent the far receiver from seeing garbage when we are no longer sending to it. It is the responsibility of software to make sure that no traffic is going across the link when EndOfChain is set, so that the switch to NOPs does not occur in the midst of a packet.

EndOfChain resets to 0. It can be set by software by writing a 1, to indicate the logical end of the chain, or partition a double-hosted chain into two independent logical chains. This bit cannot be cleared by software – a write of 0 to this bit position has no effect.

For host interface blocks, this bit is reset by LDT RESET#, not host reset. For secondary interface blocks, this bit is reset by LDT RESET# on the bridge's secondary chain, not LDT RESET# on the primary chain.

7.5.4.7 Transmitter Off (bit 7)

This bit provides a mechanism to shut off a link transmitter for power savings or EMI reduction. When set, no output signals on the link toggle. This bit resets to 0, and can be set by software writing a 1 to the bit. This bit cannot be cleared by software – a write of 0 to this bit position has no effect. If EndOfChain is set on an active link, TransmitterOff should not be set until the transmitter has driven enough NOPs to fill the receiver's receive FIFOs. When this bit is set, the link receiver should also be disabled if necessary to prevent DC current paths as a result of the inputs floating.

For host interface blocks, this bit is reset by LDT RESET#, not host reset. For secondary interface blocks, this bit is reset by LDT RESET# on the bridge's secondary chain, not LDT RESET# on the primary chain.

7.5.4.8 CRC Error (bits 11:8)

These bits are set by hardware when a CRC error is detected on an incoming link. Errors are detected and reported on a per byte-lane basis where bit 8 corresponds to the least-significant byte lane. Four bits are required to cover the maximum LDT width of 32 bits. Error bits for unimplemented (as specified by Max Link Width In, section 7.5.5.1) or unused (as specified by Link Width In, section 7.5.5.5) byte lanes return 0 when read.

These bits are persistent through warm reset, and are reset to 0 at cold reset. They are readable from software, and can be individually cleared by software by writing a 1.

7.5.4.9 Isochronous Enable (bit 12)

This optional bit controls whether isochronous flow control, as described in section D.1, is enabled for this link. The bit is set to enable ISOC mode and clear to disable ISOC mode. Note that the Isoc bit in requests and responses is used regardless of this setting. Only Isoc flow control packets are prevented by clearing it. If implemented, this bit is readable and writable from software, and persistent through warm reset. Cold reset clears the bit. To enable ISOC mode operation after a cold reset, software must set the bit and sequence the chain through a warm reset. This bit is reserved if the Isochronous Mode Capability bit is cleared. (See section 7.5.9.1) It is the responsibility of system sizing software to ensure that this bit is set to the same value on both sides of the link and only set if both sides of the link have Isochronous Mode capability.

7.5.4.10 LDTSTOP# Enable (bit 13)

This optional bit controls whether the transmitter tristates the link during the disconnected state of an LDTSTOP# sequence, as described in section 8.5. When the bit is set, the transmitter tristates the link, when the bit is clear it continues to drive the link. If implemented, this bit is cleared by cold reset and persistent through warm reset. This bit is reserved if the LDTSTOP Capability bit is cleared. (See section 7.5.9.2) The behavior of the link transmitter and receiver in both the tristate and driven cases is described in the table below:

LDTSTOP# Enable	Link State in LDTSTOP# disconnect state	Transmitter behavior	Receiver behavior
0	Driven	CAD, CTL & CLK logically undefined, but driven to electrical levels which satisfy DC specification.	Ignores CAD, CTL & CLK logical values.
1	Tristate	CAD, CTL & CLK placed in high impedance state.	Disables DC current paths that could be created as a result of CAD, CTL & CLK inputs being tristated and ignores logical values.

7.5.5 Link Config Register: Offsets 06h & 0Ah

As with the Link Control Register, there may be either one or two copies of this register, one for each link. If only one link is implemented by the device, the second register is reserved. All unspecified bits are reserved.

As described in the following subsections, software updates to the upper half of this register take effect after a warm reset sequence and, depending on the field, also after an LDTSTOP# disconnect sequence. For host interface blocks, the change is affected by LDT RESET#, not host reset. For secondary interface blocks, the change is affected by LDT RESET# on the bridge's secondary chain, not LDT RESET# on the primary chain.

15	14 12	11	10 8	7	6 4	3	2 0
DwFcOutEn	LinkWidthOut	DwFcInEn	LinkWidthIn	DwDcOut	MaxLinkWidthOut	DwFcIn	MaxLinkWidthIn

7.5.5.1 Max Link Width In (bits 2:0)

This field contains three bits that indicate the physical width of the incoming side of the LDT link implemented by this device. It is read-only.

The encodings are as follows:

LinkWidth[2:0]	Width
000	8 bits
001	16 bits
010	Reserved
011	32 bits
100	2 bits
101	4 bits
110	Reserved
111	Link physically not connected (LinkWidthIn, LinkWidthOut only)

7.5.5.2 Doubleword Flow Control In (DwFcIn, bit 3)

This bit is set to indicate that this receiver is capable of doubleword-based data buffer flow control. It is read-only.

7.5.5.3 Max Link Width Out (bits 6:4)

This field contains three bits that indicate the physical width of the outgoing side of the LDT link implemented by this device. It is read-only. It uses the same encodings as the MaxLinkWidthIn field.

7.5.5.4 Doubleword Flow Control Out (DwFcOut, bit 7)

This bit is set to indicate that this transmitter is capable of doubleword-based data buffer flow control. It is read-only.

7.5.5.5 Link Width In (bits 10:8)

This field controls the utilized width (which may not exceed the physical width) of the incoming side of the links of the LDT link implemented by this device. It uses the same encodings as the MaxLinkWidthIn field. It is readable and writable from software, and persistent through warm reset. After cold reset, this field is initialized by hardware based on the results of the link width negotiation sequence described in section 12.2. This sequence also identifies physically unconnected links. Based on sizing the devices at both ends of the link, software can then write a different value into the register. The chain must pass through warm reset or an LDTSTOP# disconnect sequence for the new width values to be reflected on the link.

The LinkWidthIn CSR in the link receiver must match the LinkWidthOut CSR in the link transmitter of the device on the other side of the link. The LinkWidthIn and LinkWidthOut registers within the same device are not required to have matching values.

7.5.5.6 Doubleword Flow Control In Enable (DwFcInEn, bit 11)

This bit may be set to program the receiver into doubleword-based flow control mode. At cold reset, this bit is cleared. Based on examining devices at both ends of the link, software can then write a different value into the register. The chain must pass through warm reset for the new flow control method to be used on the link.

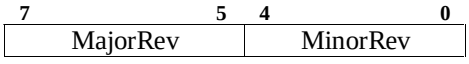
7.5.5.7 Link Width Out (bits 14:12)

This field is similar to the LinkWidthIn field, except that it controls the utilized width of the outgoing side of the links implemented by this device. Like LinkWidthIn, this field is initialized after cold reset by hardware based on the results of the link width negotiation sequence described in section 12.2. Byte lanes that are disabled due to the LinkWidthOut value being set narrower than the physically implemented width of the link will have their transmitters shut down in the same way as if TransmitterOff was set.

7.5.5.8 Doubleword Flow Control Out Enable (DwFcOutEn, bit 15)

This bit is similar to DwFcInEn, except that it puts the transmitter into doubleword-based flow control mode. It is cleared by cold reset.

7.5.6 LDT Revision ID Register: Offset 08h or 0Ch



7.5.6.1 Minor Revision (bits 4:0)

This read-only field contains the minor revision of the LDT specification that the particular implementation conforms to.

7.5.6.2 MajorRevision (bits 7:5)

This read-only field contains the major revision of the LDT specification that the particular implementation conforms to.

7.5.7 LinkFreq Register: Offsets 09h or 0Dh & 11h

As with the Link Control and Link Config registers, there may be either one or two copies of this register, one for each link. If the device only implements one link, the second register is reserved.

The LinkFreq register specifies the maximum operating frequency of the link's transmitter clock – the data rate is twice this value. The encoding of this field is shown below.

LinkFreq Encoding	Maximum Transmitter Clock Frequency (MHz)
0000	200(default)
0001	300
0010	400
0011	500
0100	600
0101	800
0110	1000
0111 to 1110	Reserved
1111	Vendor-Specific

The Link Freq Register is cleared by cold reset and its contents are persistent across warm reset. Software can write a nonzero value to this register, and that value will take effect as a result of either a warm reset or LDTSTOP# disconnect sequence. For host interface blocks, the change is affected by LDT RESET#, not host reset. For secondary interface blocks, the change is affected by LDT RESET# on the bridge's secondary chain, not LDT RESET# on the primary chain.

See section 11.1 for a definition of the LDT clocking modes, and for how the LinkFreq register controls the LDT transmitter frequency in each mode.

LDT devices are not required to support all the transmitter clock frequencies in the above table. All LDT devices must support a 200-MHz synchronous link.

7.5.8 LinkFreqCap Register: Offsets 0Ah or 0Eh & 12h

The Link Frequency Capability register (LinkFreqCap) is a 16-bit read only register that indicates the clock frequency capabilities of the associated link. Each bit in LinkFreqCap corresponds to one of the 16 possible encodings of the LinkFreq register as defined in section 7.5.7. Bit N of LinkFreqCap corresponds to encoding N of the LinkFreq field. A one in LinkFreqCap means that the link supports the corresponding link frequency, and a zero means the link does not support that frequency. Bit [0] of LinkFreqCap must be one, since all links support 200MHz operation. A one in bit [15] indicates that vendor-specific frequencies are available, the use and support of which are beyond the scope of this specification.

The read-only value in LinkFreqCap specifies the frequency capabilities of the link independent of other practical constraints. For example, a specific device with multiple LDT links may require all the links to run at the same frequency, or the system's electrical parameters may impose frequency restrictions on a specific link's operation that are not reflected in the Link FreqCap value. It is suggested (but not required) that back-door or write-once access to these bits be provided so that implementation-specific firmware can remove frequencies that are known to be unavailable due to system-specific constraints.

7.5.9 Feature Capability Register: Offset 0Ch or 10h

This register contains bits to indicate which optional features are supported by this device. All unspecified bits are reserved.

7.5.9.1 Isochronous Mode (bit 0)

This is a read-only bit that is set to indicate that the device is capable of supporting isochronous operation as defined in section D.1, and clear to indicate that the device is not. Isochronous operation is enabled by Link Control Register bit 12.

7.5.9.2 LDTSTOP# (bit1)

This is a read only bit that is set to indicate that the associated interface supports the LDTSTOP# protocol, as described in section 8.5,and clear to indicate that it does not.

7.5.9.3 CRC Test Mode (bit2)

This is a read only bit that is set to indicate that the associated interface supports the CRC Testing Mode, as described in appendix G,and clear to indicate that it does not.

8 System Management

LDT includes features that can be deployed in PC-class systems to implement x86 legacy behaviors or to implement PC system-level behaviors such as low-power state transitions. These features are also useful for power management in non-PC systems that require power management, and LDTSTOP# provides a faster method to change link frequency and width than warm reset. From an LDT I/O Link Specification perspective, support of these system management features is optional. However, all LDT devices are required to be capable of forwarding the system management ~~broadcast~~ packets.

LDT system management supports several system-level functions. This section lists each of the functions and the means by which they are implemented using LDT system management messages. The term system management controller (SMC) is used in this section to denote the LDT device which controls system management state transition and legacy x86 pin sequencing.

8.1 Command Mapping

The SMC generates upstream system management requests by directing a posted byte WrSized packet to the system management address range defined in section 9. The count field is always 0, which indicates that only a single doubleword data packet follows the write, and it contains byte masks, not data. The byte masks are not used by the system management request and must always be all 0 bits. The format of these packets is shown below:

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Count[1:0]		Rsv	SrcTag[4:0]				
3	Rsv						Count[3:2]	
4	SysMgtCmd[7:0]							
5	Addr[23:20]				Rsv			
6	Addr[31:24]							
7	Addr[39:32]							

The host generates downstream system management requests by sending a posted broadcast packet down all the LDT I/O chains in the system. The address range in the broadcast packet identifies it as a system management request. The format of this packet is shown below:

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				

Bit Time	7	6	5	4	3	2	1	0
2	Rsv			SrcTag[4:0]				
3	Rsv							
4	SysMgtCmd[7:0]							
5	Addr[23:20]				Rsv			
6	Addr[31:24]							
7	Addr[39:32]							

For both upstream and downstream cases, the type of system management request (SysMgtCmd[7:0]) is encoded as follows:

SysMgtCmd	Command Type
0000 xxxx	Reserved
0001 xxxx	X86 legacy inputs to the processor. New state of signal: [0]: IGNNE [1]: A20M [2]: Reserved [3]: Reserved
0010 xxxx	X86 legacy output from the processor. New state of signal: [0]: FERR [3:1]: Reserved
0011 xxxx	[0]: STPCLK. Bits [3:1] are reserved for platform-specific use
0100 xxxx	SHUTDOWN Bits [3:0] reserved for host-specific use.
0101 xxxx	HALT Bits [3:0] reserved for host-specific use.
0110 xxxx	STOP_GRANT Bits [3:0] reserved for host-specific use.
0111 xxxx	VID/FID Change Bits [3:0] reserved for host-specific use.
1000 xxxx	WBINVD Bits [3:0] reserved for host-specific use.
1001 xxxx	INVD Bits [3:0] reserved for host-specific use.
1010 xxxx	[0]: SMIACK Bits [3:1] are reserved for platform-specific use
1011 xxxx	X86 platform-specific functions.
11xx xxxx	Reserved

8.2 x86 Legacy Signals: Inputs to the Processor

The information associated with the x86 legacy signals is transported using system management packets in LDT systems. The legacy signals that are inputs to processors are as follows:

- IGNNE
- A20M
- STPCLK

These packets originate from the SMC and are sent upstream to the host. For each bit, a one represents an assertion of the associated legacy pin, and a zero represents a deassertion of that pin.

8.3 x86 Legacy Signals: Outputs from the Processor

The legacy signals that are outputs from processors are as follows:

- FERR
- SMIACK

These packets originate from the host and are broadcast downstream to all LDT I/O devices in the system. For each bit, a one represents an assertion of the associated legacy pin, and a zero represents a deassertion of that pin.

The legacy pin represented by SMIACK is asserted when the processor enters system management mode (SMM) and is deasserted when the processor exits SMM.

8.4 Special Cycles

The special cycles carried by system management packets are as follows:

- HALT: generated by processor in response to execution of a HALT instruction
- SHUTDOWN: generated by processor in response to a catastrophic error
- STOP_GRANT: generated by processor in response to a STPCLK assertion
- VID/FID Change: generated by processor in response to a software controlled voltage (VID) or frequency (FID) change
- WBINVD: generated by processor in response to execution of a WBINVD instruction
- INVD: generated by processor in response to execution of an INVD instruction

These packets originate from the host and are broadcast downstream to all LDT I/O devices in the system.

8.5 Disconnecting and Reconnecting LDT Links

The LDT I/O Link specification comprehends the need for system states in which the LDT links are disabled for power savings reasons, and therefore includes two features to facilitate this behavior. While these features have been described elsewhere in the specification, this section defines their use. These features are:

1. The disconnect form of the NOP packet
2. The LDTSTOP# signal.

The system-level conditions that control the assertion and deassertion of LDTSTOP# are outside the scope of this specification. However, here are the rules that govern the use of LDTSTOP# and the disconnection and reconnection of the LDT link.

1. Once LDTSTOP# is asserted, it must remain asserted for at least 1 us. LDTSTOP# assertion must not occur while new link frequency and width values are being assigned by link sizing software or undefined operation may occur.
2. PWROK and RESET# assertions have priority over LDTSTOP# assertion, and LDTSTOP must be deasserted before RESET# is deasserted. See section 12.2 for more details.
3. An LDT transmitter that perceives the assertion of LDTSTOP# finishes sending any control packet that is in progress and then sends a disconnect NOP packet (bit [6] in the first bit-time set). After sending this packet the transmitter continues to send disconnect NOP packets through the end of the current CRC window (if the window is incomplete) and continuing through the transmission of the CRC bits for the current window. After sending the CRC bits for the current window the transmitter continues to drive disconnect NOP packets on the link for no less than 64 bit times ~~and no more than 128 bit times (independent of link width)~~, after which point the transmitter waits for the corresponding receiver on the same device to complete its disconnect sequence then disables its drivers. No CRC bits are transmitted for the last (partial) CRC window, which only contains disconnect NOP packets. Since the LDT protocol allows control packets to be inserted in the middle of data packets, and since LDT

transmitters react to the assertion of LDTSTOP# on control packet boundaries, a given data packet could be distributed amongst 2 or more devices after the LDTSTOP# disconnect sequence is complete.

4. An LDT receiver that receives the disconnect NOP packet continues to operate through the end of the current CRC window and into the next CRC window until it receives the CRC bits for the current window. After sampling the CRC bits for the current window the receiver disables its input receivers.
5. Note that LDTSTOP# can deassert either before or after the link disconnection sequence is complete. A link transmitter is not sensitive to the deassertion of LDTSTOP# until both its disconnect sequence as described in step 2 is complete, and the disconnect sequence for the associated receiver on the same device is complete. A link receiver is not sensitive to the deassertion of LDTSTOP# until both its disconnect sequence is complete, and the disconnect sequence for the associated transmitter on the same device is complete.
6. An LDT transmitter that perceives and is sensitive to the deassertion of LDTSTOP# immediately enables its drivers, begins toggling the LDT clock, and places the link in the state associated with the beginning of a warm reset sequence (CTL = 0. CAD = 1's). An LDT receiver that perceives and is sensitive to the deassertion of LDTSTOP# waits for at least 1 us and then enables its input receivers. When a transmitter's corresponding receiver on the same device has been enabled, it is free to begin the post warm-reset initialization sequence described in section 12.2. This 1 us delay is required to prevent a device from enabling its input receivers before the transmitter on the other side of the link has perceived and reacted to the deassertion of LDTSTOP#. ~~This consideration is also the reason that, as described in step 2, the transmitter must disable its drivers and enter the disconnected state relatively quickly (no more than 128 bit times after sending the CRC bits for the last full CRC window).~~
7. After reconnecting to the link, the first transmitted packet after the warm reset initialization sequence must be a control packet, as implied by the state transitions of the CTL wire during link initialization. This is true even if the link was disconnected in the middle of a data packet transmission.
8. The CRC logic on either side of the link should be re-initialized after a disconnect sequence in exactly the same way as for a reset sequence.
9. LDT link disconnect and reconnect sequences do not cause flow control buffers to be flushed, nor do they cause flow control buffer counts to be reset.

The electrical state of the LDT link during the LDTSTOP# disconnect state is controlled by a configuration bit, as described in section 7.5.4.10.

9 ~~System~~ LDT Address Map

Base Address	Top Address	Size	Use
00_0000_0000h	FC_FFFF_FFFFh	~~1,008~~1012 Gbytes	DRAM / Memory-Mapped I/O
FD_0000_0000h	FD_F7FF_FFFFh	3,968 Mbytes	Reserved
FD_F800_0000h	FD_F8FF_FFFFh	16 Mbytes	Interrupts / LDT EOI
FD_F900_0000h	FD_F90F_FFFFh	1 Mbyte	Legacy PIC IACK
FD_F910_0000h	FD_F91F_FFFFh	1 Mbyte	System Management
FD_F920_0000h	FD_FBFF_FFFFh	46 Mbytes	Reserved
FD_FC00_0000h	FD_FDFF_FFFFh	32 Mbytes	~~PCI~~ I/O
FD_FE00_0000h	FD_FFFF_FFFFh	32 Mbytes	~~PCI~~ Configuration
FE_0000_0000h	FF_FFFF_FFFFh	8 Gbytes	Reserved

The bulk of the address space can be used for either memory or memory-mapped I/O. The partitioning of this space into regions for each is implementation-specific.

10 Error Handling

10.1 Error Conditions

10.1.1 Transmission Errors: 8-Bit, 16-Bit, and 32-Bit Links

A 32-bit cyclic redundancy code (CRC) covers all LDT links. The CRC is calculated on each 8-bit lane independently and covers the link as a whole, not individual packets. CTL is included in the CRC calculation. In each bit time, CAD is operated on first, beginning with bit 0, followed by CTL. For 16-bit and 32-bit links, where the upper byte lanes do not have a CTL bit associated with them, a CTL value of 0 (Data) is used.

The CRC is computed over 512 bit times. Each new CRC value is stuffed onto the CAD bits of the link 64 bit times after the end of the 512-bit-time window, and occupies the link for 4 bit times. Therefore, bit times 64-67 (the first bit time being 0) of each CRC window after the first contain the CRC value for the previous window. There is no CRC transmission during the first 512-bit-time window after the link is initialized, and the value of the transmitted CRC bits is not included in the CRC calculation for the current window. Therefore each CRC window after the first is 516 bit times in length—512 of which are included in the calculation of the CRC which will be transmitted in the next window. There is no indication on the bus that CRC information is being transmitted. It is the responsibility of the parties on both ends of the link to count cycles from the beginning of the first valid packet after link synchronization to determine the boundaries of the CRC windows. During transmission of the CRC, the CTL bit will be driven to a value of 1 (Control).

For example, the contents of 8,16 and 32-bit links during the first three CRC windows after link synchronization are shown below:

CRC Window after sync	Number of Bit times	Link contains
1st	512	Payload for 1st window
2nd	64	Payload for 2nd window
	4	CRC of 1st window
	448	Payload for 2nd window
3rd	64	Payload for 3rd window
	4	CRC for 2nd wondow
	448	Payload for 3rd window

The polynomial used to generate the CRC is:
$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.

The CRC is calculated by computing the remainder resulting from the division of the data by the CRC polynomial. The register used to perform the calculation is seeded with all 1 bits at the beginning of each CRC window. Note that in the classical CRC definition, thirty-two 0 bits are appended to the end of the data word before performing the division. This is not done in LDT. The CRC bits are inverted before being transmitted on the link in order to catch a wider range of bit errors.

The code below shows the calculation performed on the CRC accumulation register across a single bit-time (9 bits) of data.

```
static uint poly = 0x04C11DB7;          /* the polynomial */

uint compute_crc(uint data, uint crc)
{
  int i;
  for (i=0; i<9; ++i) {
    uint tmp = crc >> 31;               /* store highest bit */
    crc = (crc << 1) | ((data >> i) & 1);   /* shift message in */
    crc = (tmp) ? crc ^ poly : crc;     /* substract poly if greater */
  };
  return crc;
```

};

All link errors, once detected, are fatal, and must be assumed to have corrupted both data and control information. If the CRC Flood Enable CSR bit is set, the detecting node will drop all operations in progress, and drive Sync packets continuously on all outgoing links. It will ignore all incoming packets from the fabric. It will also set the corresponding Link Failure CSR bit.

10.1.2 Transmission Errors: 2-Bit and 4-Bit Links

For the purpose of CRC coverage, 2-bit and 4-bit links are analogous to 8-bit links running at quarter and half speed, respectively. That is, the CRC value generated for 2-bit or 4-bit links is identical to that generated for an 8-bit link carrying the same values. The extra CTL values aren't used by the receiver and aren't included in the CRC calculation.

The table below summarizes the CRC differences between 2-bit, 4-bit, and 8-bit (or wider) links.

What	**8-Bit**	**4-Bit**	**2-Bit**
CRC calculation (LSB first) ("||" means concatenation)	CTL || CAD[7:0]	CTL_0 || CAD_1[3:0] || CAD_0[3:0]	CTL_0 || CAD_3[1:0] || CAD_2[1:0] || CAD_1[1:0] || CAD_0[1:0]
CRC window size	512 bit times	1024 bit times	2048 bit times
CRC value stuffed onto CAD:	64 bit times after start of window	128 bit times after start of window	256 bit times after start of window
CRC transmission length	4 bit times	8 bit times	16 bit times
CRC test mode duration	512 bit times	1024 bit times	2048 bit times

10.1.3 Response Errors

Nodes generating TgtDone and RdResponse packets can set an error bit in these responses when an internal error condition occurs that may be localized to the transaction that the response packet is part of. The conditions that would cause this are beyond the scope of this specification. Response errors are also generated when a non-posted request traverses the entire LDT chain without being accepted by any unit.

When an error bit is set in a response, the transaction that the response was a part of is considered to have failed. If the transaction was a read, the returning data cannot be used. If the transaction was a write, the target location must be assumed to have gone to an undefined state.

In all cases, the goal is to complete the transaction protocol in the fabric, while returning the error notification to the requester. If the response is being returned to the requester, the latter is accomplished. The requester issues any handshakes necessary to complete the transaction.

RdResponse packets with the error bit set still transmit the expected quantity of data, even though it cannot be used by the destination.

10.2 Error Handling

Once detected, errors can cause one of several actions (in order of severity):

1. Signal requester via response error bit. It is then the responsibility of the requester to determine the appropriate action. This is only appropriate for errors that can be localized to a single transaction, which can be completed and retired without hanging any fabric resources.
2. The error detector can generate an NMI to a processor.

3. The error detector can broadcast Sync packets. This will bring the fabric down and require a hard reset to reinitialize the fabric.

10.3 Sync Packets

Sync packets are used to broadcast fatal error information within all fabrics. Nodes detecting fatal errors drive their outgoing CAD and CTL bits on all links to logic 1, and keep them there. This is recognized by devices at the other ends of the links as a sync packet, even if the nodes are out of sync or the clock has been corrupted. The node driving the error drops all transactions currently in flight and ignores any packets received from its incoming links. CRC is not generated on links transmitting sync packets, nor is it checked on incoming links on which a sync packet has been detected.

Nodes receiving sync packets behave in exactly the same fashion as nodes detecting fatal errors, and they propagate the error out all of their outgoing links. As sync packets propagate around the system, nodes will shut down, and activity in the fabric will cease. This state is stable, and will persist until a warm reset occurs.

11 Clocking

LDT systems consist of devices connected by LDT links. Devices within an LDT fabric may or may not be clocked by clocks derived from the same frequency source. Section 11.1 describes the requirements for LDT device clock sources.

11.1 Clocking mode definitions

Each LDT device has a transmit clock, which is used to generate its LDT clock outputs, and a receive clock, to which incoming packets are synchronized in the receiver.

Three operation modes of LDT devices are defined.

In **synchronous (sync)** mode, each transmit clock must be derived from the same time base as the receive clock in the device to which it is connected. In addition, the LDT output clocks from each device must operate at the same frequency – that being the frequency programmed by their respective LinkFreq registers.

In **pseudo-synchronous (pseudo-sync)** mode, each transmit clock must be derived from the same time base as the receive clock in the device to which it is connected. The LDT output clock frequency for one device may be both arbitrarily lower than the frequency programmed into its LinkFreq register, and arbitrarily lower than the receive clock frequency in the other device. The LDT output clock frequency for one device must not exceed the receive clock frequency in the other device.

In **asynchronous (async**) mode, each transmit clock need not be derived from the same time base as the receive clock in the device to which it is connected. In order to cope with frequency error due to running nominally matched transmitter/receiver pairs from different time bases, the maximum LDT output clock frequency for one device can exceed the maximum receive clock frequency in the other device by no more than 2000 parts per million (2000 ppm). Further, the LDToutput clocks can exceed the frequency programmed into the LinkFreq registers by no more than 1000 ppm. As in pseudo-sync mode, the LDT output clock frequency for one device may be both arbitrarily lower than the frequency programmed into its LinkFreq register, and arbitrarily lower than the receive clock frequency in the other device. See section 11.3 for a description of one scheme for handling this situation.

See section 7.5.7 for a description of LDT link frequency selection.

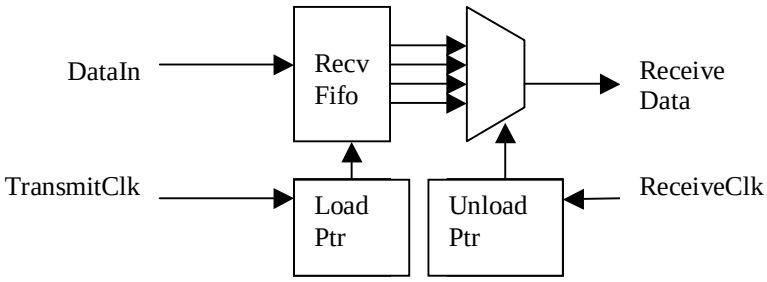
All LDT devices must support sync mode operation. Devices may also implement pseudo-sync and async modes based on their unique requirements. The means by which the operating mode is selected for a device that can support multiple modes is outside the scope of this specification.

11.2 Receive FIFO

Each LDT receiver contains a FIFO that is clocked by the transmitter. See the LDT electrical specification for the details of this structure and the guidelines that govern its design. This section introduces the receive FIFO only to motivate the discussion in sections 11.3 and 12.2.

The FIFO is sized to absorb the *dynamic variation* between the transmitter's clock and the receiver's clock. Some sources of this dynamic variation are:

- Temperature
- Voltage (either of the transmitter or the receiver)
- Accumulated phase error in a PLL
- Noise which affects the clock and data in the same way. (If the noise affected clock and data differently then this would affect the maximum bit rate, not the buffer depth.)



For links that are wider than 8 bits, the FIFO absorbs the difference in delay between 8-bit segments of the link. For links in which the two connected devices are clocked by different sources, the FIFO absorbs the frequency variation. See section 11.3. Also, the FIFO can be used to provide buffering between a narrow high-speed link and a wider slower datapath inside a receiver.

Clocking successive bit-times into different FIFO entry flops serves to increase the valid time of each flop. Before operating the link, the load and unload pointers must be initialized relative to each other. The LDT protocol that supports this requirement is described in section 12.2.

11.3 An Async Mode Implementation Example

Since in async mode the transmit clock in one device and the receive clock in the other device are not derived from the same time base, the relationship between the FIFO load pointer and unload pointer may have to be adjusted dynamically.

If the receive clock is faster than the transmit clock, then the unload pointer occasionally has to be frozen. This situation can occur in both pseudo-sync and async modes.

If the receive clock is slower than the transmit clock, one method for ensuring that the transmitter does not overrun the receiver is described below. Note that this situation can only occur in async mode. The transmitter frequency can exceed its programmed frequency by no more than 1000 ppm, and can exceed the receiver's frequency by no more than 2000 ppm. This is less than the rate at which CRC bits are inserted onto the link (4 parts in 1092 for the first CRC window after initialization). CRC bits are sent by the transmitter and recomputed by the receiver. The receiver recomputes the CRC bits from the packet stream that is registered into the receiver's clock domain from the receive FIFO. The CRC bits are not placed directly into the receive FIFO, however. Instead, the CRC bits are placed into dedicated flops which are

clocked by the transmit clock, and the bits are evaluated by receive clock logic only after sufficient time has passed to ensure that these flops can be reliably sampled. Since the CRC bits only appear every 512 bit times there is a huge sample window for these flops. By not placing CRC bits into the receive FIFO and therefore not incrementing the unload pointer, the receiver can always catch up to the transmitter.

11.4 Link Frequency Initialization and Selection

Cold reset initializes LDT I/O link transmitters to a link clock frequency of 200 MHz. All LDT I/O link receivers must support an LDT clock of at least 200 MHz. Initialization firmware can reprogram the link transmitter frequencies and initiate a warm reset or LDTSTOP# disconnect sequence to invoke the change to the link clock frequencies. See section 7.5.7 for details.

12 Reset and Initialization

12.1 Definition of Reset

Two types of reset are defined at the fabric level as follows:

1. Cold Reset – Node logic is reset. All links are reset. UnitIDs are assigned. All CSRs are reset. Cold reset is caused by the deassertion of PWROK together with the assertion of RESET#. Note that this sequence may be initiated under software control.
2. Warm Reset – Same as Cold Reset, except that CSRs defined to be persistent (expected to be mostly error state) are not reset. Warm reset is caused by asserting RESET# and keeping PWROK asserted. It may be initiated under software control.

The means by which PWROK and RESET# are generated within a specific system are outside of the scope of this specification.

12.2 System Powerup, Reset, and Low-Level Link Initialization

For a cold reset sequence, PWROK is asserted at least 1 ms after the power and clock sources for all LDT devices have become stable. RESET# must be asserted 1ms before PWROK is asserted, and RESET# must remain asserted for at least 1 ms beyond the assertion of PWROK. Since the state of RESET# is undefined during some of the time before PWROK is asserted, PWROK's deassertion should be combined with RESET# to generate internal resets.

For a warm reset sequence, RESET# must be asserted for at least 1 ms.

LDTSTOP# must be deasserted at least 1 us before RESET# is deasserted, and it must remain deasserted until the link has completed the synchronization sequence described below.

A cold reset enables transactions to flow across the link using the minimum CAD widths that are common to the transmitters and receivers on each side of the link. This is accomplished as follows:

An LDT component whose LDT receiver is connected to a narrower transmitter on another LDT component must have its unused CAD inputs connected to a logical zero. An LDT component whose LDT link is not used in the system must have its CLK, CTL and CAD inputs connected to a logical zero.

While RESET# is asserted during a cold reset, each LDT link controller drives its CTL signal to a logical zero, and drives its output CAD signals to a value that is based on the width of its receiver, according to the table below. If the transmitter is narrower than the receiver, all the output CAD signals are driven to a logical one.

Receiver width (bits)	CAD[31:0] value driven	
2	0000 0003	

4	0000 000F	
8	FFFF FFFF	Req'd for backward compatibility
16	FFFF FFFF	Req'd for backward compatibility
32	FFFF FFFF	

At the deasserting edge of RESET#, each LDT controller samples its input CAD signals and uses this sampled value to determine its transmitter and receiver widths, according to the table below. The result of this process is reflected in the cold reset values of the LinkWidthIn and LinkWidthOut registers.

CAD[31:0] value sampled (hex)	Transmitter & Receiver widths (bits)	
0000 0000	N/A	Unused link.
0000 0003	2	
0000 000F	MIN(4, Receiver width, Transmitter width)	
0000 00FF	MIN(8, Receiver width, Transmitter width)	
0000 FFFF	MIN(8, Receiver width, Transmitter width)	Req'd for backward compatibility
FFFF FFFF	MIN(8, Receiver width, Transmitter width)	Req'd for backward compatibility

Warm reset preserves the transmitter and receiver widths programmed by software, so it is slightly different than cold reset.

While RESET# is asserted during a warm reset, each LDT device drives its outbound link(s) to the following state

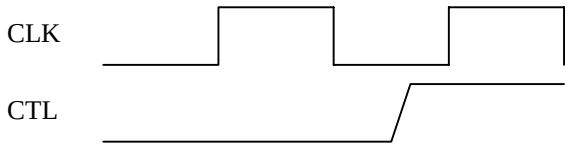
Signal	**State During Reset**
CLK	Toggling
CTL	Logic 0
CAD[1:0]	Logic 1
CAD[31:2] (if present)	Logic 1

Note that this state does not correspond to any particular LDT packet type.

The link initialization sequence from this point forward in time involves transmitting and receiving values on the CTL and CAD signals, and it is the same for cold and warm reset. The timing of the sequence described below, in terms of bit-times, is the same for 8-bit, 16-bit, and 32-bit links. The bit-time counts for 4-bit and 2-bit links should be doubled and quadrupled, respectively.

The discussion below makes reference to the CLK edges (rising or falling) across which the transmitter places new values on the CTL and CAD signals. Implied in this text is that the receiver registers the new data (using the transmitted clock) using that the same clock edge (rising or falling). The transmitter's physical interface to the link delays the clock relative to the data in order to position the clock in the middle of the data window.

A device-specific period of time after the deassertion of RESET#, each device asserts its CTL signal across a rising CLK edge, initiating a sync sequence. The timing of this transition is shown below.



The assertion of the CTL signal serves to indicate to the device at the other side of the link that this device is ready to initialize the link. Devices perform whatever device-specific functions they may require between

the time RESET# is deasserted and the time they assert CTL. This may include ramping their internal clocks to full frequency and reading configuration state from off-chip.

When a device has sampled the assertion of both its own CTL wire and the CTL wire driven from the other device, it drives the Sync packet for at least 16 more bit times and then inverts both CAD and CTL across a rising clock edge.

The deassertion of the incoming CTL/CAD signals across a rising CLK edge is used in the transmit clock domain within each receiver to initialize the load pointer. The deassertion of the incoming CTL and CAD signals is synchronized to the core clock domain and used to initialize the unload pointer within each receiver. The length and uncertainty of this synchronizer must be included to determine the proper relationship between the load pointer and the unload pointer. Note that CTL cannot be used to initialize the pointers for byte lanes other than 0 in a multi-byte link, as CTL only exists within the byte 0 transmit clock domain. After this point, all transitions of CTL must be on a 4-byte boundary

Each device continues to drive this state on its outbound links for the number of bit times shown in the table below ~~(512, 1024 or 2048 depending on the link width)~~.

Each device then drives the CAD lines to logic 1 on a 4-byte boundary across a rising CLK edge, while leaving the CTL line deasserted, for exactly four bit times. The transition from all CAD lines deasserted to all CAD lines asserted serves to frame incoming packets. The first bit time after these four must have CTL asserted, and is both the first bit time of a new command packet and the first bit time of the first CRC window. It also occurs across a rising CLK edge.

The entire sequence is shown in the table below.

CTL	CAD	Duration: 8, 16, and 32-bit Links (Bit times)	Duration: 2 and 4-bit Links (Bit Times)	Notes
0	1	N/A	N/A	Value held during reset
1	1	16 (minimum)	64/32 (minimum)	• CTL asserts device-specific time after RESET# deasserts • Pattern held at least N bit times after both devices sample assertion of both CTL wires.
0	0	512+4N	2048+16N/1024+8N	• 1->0 transition on incoming CTL/CAD initializes load pointer in transmit clock time domain • 1->0 transition on incoming CTL/CAD synchronized to core clock and used to initialize unload pointer in receive clock time domain
0	1	4	16/8	0->1 transition on CAD serves to frame incoming packets
1	??	N/A	N/A	0->1 transition on CTL defines start of first control packet and represents first bit time of first CRC window

Using the initialization sequence as defined in the above section, sync, pseudo-sync and async devices can inter-operate as long as they share a common input clock.

12.3 I/O Fabric Initialization

I/O fabric initialization is largely software driven. A sample initialization sequence consists of the following steps:

1. One or more reset sequence initiators assert RESET# (and possibly deassert PWROK). Each initiator must sequence the PWROK and RESET# signals according to the rules defined in section 12.2. Multiple initiators may or may not release (deassert) RESET# at the same time. In any event, the last initiator to release RESET# determines when the initialization sequence begins. Note this means that each initiator must sample as well as drive RESET#.

2. The low level link initialization sequence described in section 12.2 takes place between each node in the fabric.

3. Each node sends buffer-release packets to inform the transmitter(s) to which it connects how many buffers it contains.

In a double hosted chain, the host bridge at one end of the chain is designated the master host bridge, and the other the slave. How a host bridge determines whether it is a master or slave is outside the scope of this spec.

4. The slave host bridge, if any, sets a timer, and goes to sleep. The duration of the timer is implementation-specific. The master host bridge proceeds with the initialization sequence. At the beginning of the sequence, nextFreeID = 01h.

5. The master host bridge checks the Initialization Complete bit for its outgoing link, to determine if a device has been detected at the other end of the link. It also checks the CRC Error bits to see if the link has taken any errors since reset. If there is no device, or the link is taking errors, chain sizing is complete, proceed to step 8.

6. The bridge issues CSR accesses to Device ID 00h, which is the Base Unit ID that all devices assume at reset, and which is also the default responded to by host bridges. The reads are responded to by the first uninitialized device on the chain. Performing a write to the command register (without changing any fields) will cause the Master Host bit to get set, which indicates which link on that device is pointing toward the host bridge. By polling the CRC error bits for that link, software can determine that the device is not seeing CRC errors on the link between it and the host. Software can then set the CRC Fatal Enable bit for the link from both ends. If the link is taking CRC errors, chain sizing is complete, proceed to step 8.

7. By reading the Class Code, Vendor ID, and Device ID, software can determine what type of device it is talking to. If it has reached an LDT device, it writes the Base Unit ID with nextFreeID, and increments nextFreeID by the Unit Count value of the device. Now that the device will no longer respond to accesses to 00h, the process can be repeated for the next link, starting back at step 5. If a slave host bridge has been reached, software sets the DoubleEnded bit on both that bridge and the master host bridge, and proceeds to step 9 for link partitioning.

8. At this point, an end to the chain has been found without reaching another host bridge. Software sets the EndOfChain and TransmitOff bits for the last link in the chain, and fabric initialization is complete. If there is a bridge at the other end and the sizing algorithm hasn't reached it due to a break in the chain, it will wake up when its timer expires, find its DoubleEnded bit clear, and size the chain from the other end to the break, starting at step 5. The final result will be two single-ended chains, each with a master host bridge.

9. At this point, the entire chain has been sized, and found to have host bridges at both ends. When the slave host bridge's timer expires, it will find its DoubleEnded bit set and know that no sizing is required on its part. All intermediate devices will have their Master Host bit pointing towards the master host bridge. However, it is expected that software will wish to partition the devices in the chain between the two host bridges for load balancing reasons. If so, software must select the location at which it wishes to break the chain and then access the nodes on either side of the break from the host

bridge on that side. First, the EndOfChain bit for the link to be broken is set from each side. When both devices are ignoring the link, the TransmitOff bit for each side can be set.

The initialization process can be made more robust by providing a facility to time out the CSR accesses used for sizing, in the event that a device fails to respond. This possibility is beyond the scope of this specification.

12.3.1 Finding the firmware ROM

System implementations can be built in which software initialization code is stored in a firmware ROM that resides behind a default bridge on one of several LDT I/O chains connected to the host. Further, system implementations can be built that do not require the host to be hardware-configured to identify the I/O chain that contains the default bridge. One possible method of initializing a system with these two characteristics involves a host that after reset sends a firmware code fetch down each I/O chain connected to it. The I/O chain that contains the default bridge will respond without error, while all the other I/O chains will respond with an NXA error. This allows the host to identify the compatibility I/O chain, and subsequent firmware fetches can be directed only down that chain.

12.4 Link Width Initialization

Note that the hardware-sequenced link width negotiation sequence described in section 12.2 does not result in the links operating at their maximum width potential. 16-bit, 32-bit, and asymmetrically sized operation must be enabled by a software initialization step. Each link controller contains two pairs of control register fields relating to the width of the link as follows:

- A pair of fields that are hard-wired to indicate the maximum supported widths of the inbound and outbound links.
- A pair of fields that are initialized after a cold reset to a particular value based on the result of the link width negotiation sequence described in section 12.2. This pair of fields controls the actual link width and is persistent across a warm reset.

At cold reset all links power-up and synchronize as described in section 12.2. Firmware (known as BIOS in the x86 world) interrogates all the links in the system, reprograms all the links to the desired width, and then takes the system through a warm reset to change the link widths. See section 7.5.4.9 for details on the link width control register.

After an LDT disconnect-reconnect sequence, devices that implement the LDTSTOP# protocol described in section 8.5 are required to update their link widths in exactly the same way as they do after a warm reset sequence. This allows initialization software for systems built from such devices to use the LDTSTOP# protocol rather than warm reset to invoke link width changes.

A LDT I/O Link Fairness Policy and Algorithm

A.1 Policy

Each node is allowed to insert packets into a busy link at a rate matching that of the heaviest node forwarding traffic through it. In addition, the node can freely use any idle time on the link. This property must be met over a window in time small enough to be responsive to the dynamic traffic patterns, yet large enough to be statistically convergent. In order for a system of nodes to behave consistently, each node should implement this policy using the same algorithm as described below.

A.2 Algorithm

The algorithm consists of two parts. The first is the method used to calculate the insertion rate the node can use. The second governs how the node achieves that insertion rate. This algorithm must be implemented independently for both the upstream and downstream direction to support double-hosted chain

configurations. The algorithm requires no dedicated control or status registers and has no configurable parameters.

To calculate the insertion rate, the maximum packet rate must be deduced for the heaviest downstream node. This is done by implementing 32 three-bit counters, one for each potential downstream node as well as a single eight-bit counter. At reset all counters are reset to 0. When a request is forwarded the three-bit counter corresponding to the node that generated the packet is incremented. The eight-bit counter is incremented once for every forwarded packet. When one of the three-bit counters overflows, the value of the eight-bit counter (post increment) is captured (hereafter referred to as the denominator). All counters are then cleared. The request rate of the worst case downstream node has now been calculated and is equal to 8/denominator. On average the node can insert eight requests for every 'denominator' requests forwarded. This insertion should be paced and not inserted as bursts. Packets can always be inserted when there are no packets waiting to be forwarded. The denominator register is set to 1 on reset.

To insert, the node has an eight-bit counter referred to as Window, which at reset is set to 1. It also has a one-bit register, referred to as Priority, that at reset is cleared to 0. When a node has requests ready to be sent on the outbound links, it decides which to send based on the following cases:

1. Forward packet to send—no local packet to send:
   The forward packet is sent and the Window register is decremented.

2. No forward packet to send—local packet to send:
   The local packet is sent and the Priority register is cleared.

3. Both forward packet and local packet to send:
   If the bit in the Priority register is set, the local packet is sent and the Priority bit is cleared. Otherwise, the forward packet is sent and the Window register is decremented.

Whenever the Window register is decremented to 0, its next value is recalculated and the Priority bit is set. In order to achieve non-integral insertion rates, the new value of the Window register must be loaded probabilistically. Each node will implement a nine-bit linear feedback shift register using the polynomial x^9+x^4+1. It is advanced once every time the window register value is recalculated. The window register is loaded with (denominator+LFSR[2:0]) >> 3.

A.3 Implementation Note

Care must be taken in implementing the packet insertion logic in order to avoid a potential starvation problem. The packet inserter is basically a two-input arbiter between issued packets and forwarded packets. The requests to this arbiter are generated when there is a packet ready to go from one of these sources, and there are free buffers (as indicated by buffer release messages) at the other end of the link to receive the packet. It is possible that there is one packet to be issued and forwarded, both in the same virtual channel and therefore requiring the same buffer type(s). If the forwarded packet is chosen and there is only one buffer of the needed type free, the issued packet cannot be transmitted. When the fairness logic next allows a packet to be inserted, a packet from a different virtual channel can be chosen, allowing the priority of the packet inserter to swing back to forwarding. When the buffer release message that would allow the blocked packet to go arrives, it no longer has priority in the inserter, and therefore can't go. If another packet in the same channel is forwarded before priority changes back to inserting, this situation can persist, starving packet insertion in a particular virtual channel.

B Ordering Rules Of Supported I/O Protocols

LDT is intended to support connections to I/O bridges supporting a variety of I/O protocols. At the present time, there are two I/O bus protocols identified that we support, PCI and AGP, each of which has different ordering requirements as described below.

B.1 PCI

The PCI ordering rules are taken from the *PCI Local Bus Specification, Revision 2.2 (6/8/98 draft), Appendix E*. See that spec for more information.

Row Pass Column?	**Posted Memory Write (PMW)**	**Delayed Read Request (DRR)**	**Delayed Write Request (DWR)**	**Delayed Read Completion (DRC)**	**Delayed Write Completion (DWC)**
PMW	No	Yes	Yes	Yes	Yes
DRR	No	Yes/No	Yes/No	Yes/No	Yes/No
DWR	No	Yes/No	Yes/No	Yes/No	Yes/No
DRC	No	Yes	Yes	Yes/No	Yes/No
DWC	Yes/No	Yes	Yes	Yes/No	Yes/No

No – Indicates that the subsequently issued transaction is not allowed to complete before the previous transaction to preserve ordering in the system.

Yes – Indicates that the subsequently issued transaction must be allowed to complete before the previous transaction or deadlock may occur. Reasons for all eight *Yes* entries are given in the PCI spec. However, it indicates that the four *Yes*entries in the first row are only required for backward compatibility with earlier revisions of the spec. For the PMW/DRR and PMW/DWR cases, this is an additional reason—they are required to be *Yes* because the fourth row requires DRCs to be able to pass DRRs and DWRs. In the case where a DRR or DWR occurs followed by a PMW and followed by a DRC, the PMW must pass the DRR/DWR in order to allow the DRC to do so, because the DRC may not pass the PMW.

Yes/No – Indicates the subsequently issued transaction may be allowed to complete before the previous transaction. There are no ordering requirements between the transactions.

In addition, PCI supports the concept of performing multiple system reads due to a single PCI bus read in anticipation of that data being needed soon—this is called PCI prefetching. Within a prefetch, reads are required to be strongly ordered.

PCI is also capable of generating operations with discontiguous byte masks. If it does this for requests that cross aligned doubleword boundaries, they must be broken on doubleword boundaries into multiple transactions on LDT. In this case, the requests must be strongly ordered in ascending order by address.

B.2 AGP

These ordering rules are taken from the *Accelerated Graphics Port (AGP) Interface Specification, Revision 2.0, section 3.4*. See that spec for more information.

AGP essentially consists of three separate channels, each with its own distinct ordering rules. No ordering is maintained between the three channels—traffic is completely independent. First, AGP contains a modified PCI channel, which maintains PCI ordering. The other two channels are called the high-priority (HP) and low-priority (LP) AGP channels.

The ordering rules presented here for reads are somewhat different than what appears in the AGP spec. It defines ordering between reads in terms of the order that data is returned to the requesting device. We are concerned here with the order in which the reads are seen at the target—the I/O bridges can reorder returning read data if necessary. This leads to a slightly relaxed set of rules.

HP AGP Ordering Rules

1. Writes may not pass writes.

LP AGP Ordering Rules

1. Reads (including flushes) may not pass writes.
2. Writes may not pass writes.

3. Fences may not pass other transactions, nor be passed by other transactions.

AGP may also generate requests with discontiguous byte masks, with the same rules as PCI.

C Mapping of Protocol Ordering Rules Onto LDT

Ordering requirements for request packets are determined by their requester, and they follow the request all the way to the destination. Ordering rules for responses are also determined by the original requester and are taken from the request packet.

The sections below give the mapping from the traffic types of each of the supported protocols to LDT packet fields.

C.1 Processor

Processors should generate non-posted writes for ~~PCI~~ I/O and Configuration space. It is implementation-specific as to whether processors generate posted or nonposted writes for memory-mapped I/O.

In order to safely implement the producer-consumer model in all configurations, processor requests should follow PCI ordering rules, with the PassPW bit always clear on both requests and responses.

C.2 PCI

PCI Transaction Type	LDT Packet Type
Posted Memory Write (PMW)	WrSized, Posted, PassPW = 0
Delayed Read Request (DRR)	RdSized, PassPW = 0, RespPassPW = 0
Delayed Write Request (DWR)	WrSized, Non-posted, PassPW = 0
Delayed Read Completion (DRC)	RdResponse, PassPW = 0 (from request packet)
Delayed Write Completion (DWC)	TgtDone, PassPW = 1 *

PCI Read Line and Read Multiple requests that cause prefetches across a 64-byte boundary use sequence tagging, with each PCI Read being assigned a different non-zero sequence ID. Non-posted PCI operations with discontiguous byte masks that get broken into multiple operations on LDT must also be tagged as part of a single sequence. All other requests use a sequence ID of 0.
* In some cases, the PassPW bit must be clear in a TgtDone. See appendix F.2.4 for one example.

C.3 AGP

The three channels of AGP are all completely independent as far as ordering is concerned, so (for optimal performance) an LDT to AGP I/O bridge should assign each of these I/O streams to a separate unitID.

The PCI channel of AGP uses the PCI mapping above.

The LP and HP channels never accept requests, so there is no need to specify the ordering of returning responses with respect to requests.

HP AGP Transaction Type	LDT Packet Type
HP Write	WrSized, Posted, PassPW = 1
HP Read	RdSized, PassPW = 1, RespPassPW = 1

HP writes are placed in the posted request channel, while reads are placed in the non-posted request channel. Within each of these virtual channels, a single sequence ID is used to force the traffic to remain strongly ordered.

The PassPW and RespPassPW bits are set for reads because they are independent of the write traffic in the channel. The PassPW bit for writes does not matter in a pure HP AGP channel, because all the posted

writes in the channel are strongly ordered due to the sequence ID anyway. But, if traffic from this channel were ever mixed with another I/O stream, having it set would minimize the interaction between the two.

There are two possible mappings of LP AGP traffic onto LDT. The first puts all traffic in the LDT nonposted channel as follows:

LP AGP Transaction Type	LDT Packet Type
LP Write	WrSized, Non-Posted, PassPW = 1
LP Read	RdSized, PassPW = 1, RespPassPW = 1
LP Flush	*None*
LP Fence	*None*

All transfers in the low-priority channel are in the same virtual channel (non-posted requests), and they are all assigned to the same non-zero sequence ID, which keeps them strongly ordered. While this is a stronger ordering rule than required by AGP, it is sufficient. Since all transactions are strongly ordered, there is no need to do anything with a fence request.

Even though LP Writes are not posted in this mapping onto LDT, they can still be posted from the AGP point of view. The transaction can complete on the AGP bus without waiting for TgtDone on LDT. However, the I/O bridge must remember that TgtDone is outstanding and not retire the buffer or srcTag until it is received. Since the writes are not posted, there is also no need to issue an explicit LDT flush packet. The I/O bridge can simply wait for TgtDone to be received for all outstanding writes and then complete the flush operation on the AGP bus.

The values of the PassPW and RespPassPW bits do not matter in this mapping of a pure LP AGP channel, because there are no posted writes in this channel in either direction on LDT. However, if the traffic in this channel were ever to be combined with another I/O stream, setting them both would minimize the interactions with that stream.

The second mapping of LP AGP onto LDT puts LP writes in the posted channel:

LP AGP Transaction Type	LDT Packet Type
LP Write	WrSized, Posted, PassPW = 0
LP Read	RdSized, PassPW = 0, RespPassPW = 1
LP Flush	Flush, PassPW = 0
LP Fence	*None*

No use of nonzero sequence IDs is required. Ordering between LP writes is maintained by the fact that they are in the posted channel with their PassPW bits clear. LP reads are prevented from passing LP writes for the same reason. Flush operations use the LDT flush packet. Fences still do not result in LDT packets being sent, but they do require action in this mapping. Because no operation can pass a write, fences only need to be concerned with preventing other operations from passing reads. Therefore, they can be implemented by stalling all subsequent requests until responses have been received to all outstanding read requests.

As above, the value of RespPassPW on reads is not important in a pure LP AGP channel, but setting it may ease some interaction problems in a mixed channel.

D Considerations for Isochronous Traffic

A problem relative to bounding latency for isochronous traffic is the presence in the system of a very slow I/O device that is the target of posted requests. This situation can cause other requesters in the system to experience large and unpredictable latencies. Since LDT devices are not required to reorder responses, downstream responses with PassPW set can get stuck behind responses with PassPW clear, which are in

turn stuck behind the posted requests. In addition, peer-to-peer traffic directed at the slow device can back up the upstream posted channel as well.

The most complete (and costly) solution to this problem would be to implement ISOC mode as defined in section D.1 However, a simpler solution may be possible for most systems and is defined in section D.2.

D.1 Isochronous Flow Control Mode (optional)

In isochronous mode (ISOC mode) there are two classes of service defined. The high priority service class is intended to support isochronous traffic, and the low priority service class is intended for all other traffic. The following LDT features support the high priority service class:

- Dedicated posted command, nonposted command and response virtual channels – the ISOC virtual channels
- Dedicated flow control buffers in support of the ISOC virtual channels
- The ISOC bits in the read and write command fields identify commands that should travel in the ISOC channels.
- The ISOC bit in the read response and target done packets identify responses that should travel in the ISOC response channels
- The ISOC bit in the Nop packet identifies buffer release packets for the ISOC virtual channels.
- Broadcast, Fence, and Flush packets do not travel in or affect the ISOC virtual channels.

The following rules govern device operation in ISOC mode

1) All devices enter normal (non-ISOC) mode after a cold reset. Software may enable ISOC mode operation and sequence the chain through a warm reset to inititiate the transition to ISOC mode. See section 7.5.4.9 for details.
2) There are no ordering constraints between transactions in the ISOC and non-ISOC channels. Furthermore, ISOC traffic is invisible to the fairness algorithm implemented for non-ISOC traffic required by section 4.9.4.
3) The ordering constraints for transactions within in the ISOC channels are identical to those for transactions within the non-ISOC channel, as defined in section 6.
4) High priority traffic must always be serviced before low priority traffic, and there is no guarantee against high priority traffic starving low priority traffic, although it is expected that the total ISOC bandwidth would never exceed the overall available bandwidth. This eliminates the need for a fairness algorithm (like that in appendix A) to regulate the insertion of ISOC traffic.
5) ISOC flow control is enabled on a per-link basis to allow ISOC requests and responses to “tunnel” through non-ISOC devices on a chain.

It is intended that isochronous sources generate requests with the Isoc bit set in order to get service from the system with deterministic worst-case latency. The actual latency and bandwidth guarantee for ISOC requests is system dependent and outside the scope of this specification.

D.2 Normal Flow Control Mode

There are two observations that together may make ISOC mode unnecessary, and that can be leveraged to build simple bounded systems with more deterministic latencies:

1. No known devices use the programming paradigm that is enabled by the PCI ordering rule in which downstream responses are not allowed to pass downstream posted requests.
2. Devices that generate peer-to-peer traffic are extremely uncommon.

In light of these observations, for systems in which isochronous traffic is an important consideration, but in which the cost of ISOC mode is prohibitive, the following steps can be taken.

- Build an operating mode in which the host sets the PassPW bit in all downstream responses. This violates the PCI's downstream ordering rule but is not believed to be a material issue.
- Build LDT implementations in which responses with PassPW set will pass stalled posted requests.
- Don't populate isochronous devices on the same LDT chain as devices that generate peer-to-peer traffic.

Normal (non-ISOC) mode is characterized as follows:

- There are no dedicated ISOC virtual channels
- The Isoc bit in the Nop packet must be zero

The Isoc bit in the read, write, RdResponse, and TgtDone commands may be either set or cleared. ISOC commands may be used in normal mode in simple, bounded systems in order to get lower latency service for isochronous sources. The rules which govern the system's behavior on behalf of Isoc requests in non-ISOC mode, and the latency assurances provided to those requests are platform-specific and outside the scope of this specification. In non-ISOC mode, LDT devices may ignore the Isoc command bit, but must preserve it so that ISOC requests and responses may be handled properly by ISOC devices on either side of non-ISOC devices.

E Southbridges and Compatibility Buses

This appendix provides some considerations for including compatibility buses (such as ISA or LPC) and Southbridges in LDT-based systems.

E.1 ISA/LPC Deadlock Case

A system that contains an ISA or LPC DMA controller or supports ISA/LPC bus masters has a particular deadlock scenario that needs to be addressed. Since ISA and LPC do not support retry, a downstream posted write could block a response to a nonposted request from the ISA or LPC bus, causing deadlock.

One solution to this problem is for the host to decode programmed IO requests to the ISA/LPC memory range and emit all such requests in the nonposted channel. Alternatively the host could avoid the implemention of a positive decode for the ISA/LPC memory range and emit all default requests (those with the compat bit set) in the nonposted channel. This solution does not permit peer-to-peer requests to be issued from or to ISA/LPC devices, since such requests may result in deadlock.

E.2 ISA/LPC Write Post Flushing

Some ISA or LPC bridges require the ability to know when all posted writes they have issued are guaranteed to be globally visible. This requirement is typically handled by having a WSC# (Write Snoop Complete) pin on the Southbridge, which is asserted whenever all previously posted writes are guaranteed to be visible to all processors.

No direct LDT support is required for an LDT to PCI bridge to implement the WSC# bit. The LDT to PCI bridge can follow each posted write from the ISA/LPC bridge with an LDT flush request. The response to the flush guarantees that the posted write is globally visible. The LDT to PCI bridge can then keep its WSC# pin asserted whenever it has no flushes outstanding.

E.3 Subtractive Decoding

This section provides some considerations for including a subtractive decoding device or bridge in an LDT-based system.

E.3.1 Subtractive Decode in the General Case

Since LDT devices in a chain do not sit on the same bus, LDT devices cannot, in general, perform subtractive decode by waiting to see what requests are not responded to by other devices on the chain. Subtractive decoding devices and bridges are supported on LDT through the use of the Compat bit. All

hosts connecting to LDT I/O chains are required to have registers that specify positive decode ranges for all LDT I/O devices and bridges. One of these I/O chains may also include a subtractive bridge (potentially to a PCI, ISA, or LPC bus). Requests that do not match any of the positive ranges are routed to the LDT I/O chain containing the subtractive bridge (the *compatibility chain*) with the Compat bit set. The Compat bit indicates to the subtractive bridge that it should claim the request, regardless of address. Requests that are within the positively decoded ranges of the compatibility LDT I/O chain do not have the Compat bit set and are passed down the chain to be detected by positively decoded devices and bridges like any other LDT I/O chain.

It is worth noting that a system which sets up the subtractive decode path in hardware can talk over LDT to memory and I/O spaces owned by the subtractive device without requiring software initialization of the link. This is true even though the devices on the LDT chain have not had their unitIDs programmed to nonzero values—the Compat bit will cause accesses to reach the subtractive device.

E.3.2 Subtractive Decode in x86 Legacy Systems

Some x86 systems may have legacy software considerations that require the compatibility chain to be numbered as ~~PCI~~ Configuration Bus Number 0. In such a system the host bridge that controls the compatibility chain must be identified with a device configuration header rather than a bridge header. This bridge still requires a positive decode range so that it knows whether to set the Compat bit for transactions that do not fall in this (or any other) range. Therefore implementation-specific range registers need to be defined for this bridge.

E.3.3 Subtractive Decode in the Simplest Case

Another way to support subtractive decode in small LDT based systems is to place the subtractive decode device on the end of a single-hosted LDT chain. In that case, the subtractive device can safely assume that all requests that reach it are destined for it.

E.3.4 Subtractive Decode Behind a PCI Bridge

If the subtractive target on the LDT chain is a bridge to PCI, there are several additional issues.

The most straightforward approach is to build an LDT to PCI bridge which performs subtractive decode and implements a standard PCI bridge header. See the *PCI to PCI Bridge Architecture Specification, Revision 1.1*, for a description of subtractive decoding PCI to PCI bridges.

- The bridge performs subtractive decode using the Compat bit, as described above, for transactions which originate on the primary bus.
- The bridge performs positive decode for transactions which originate on the secondary bus. It forwards to its primary bus any transaction that originates on the secondary bus and does not fall inside the address ranges programmed into the bridge header. This implies that peer-to-peer transactions targeted at a subtractive decoding device on the secondary bus and sourced on either the secondary or subordinate buses are not supported
- The secondary bus segment is by definition not bus 0, because configuration software will encounter a bridge header and number the bus accordingly. This may not be compatible with some legacy software requirements.

Some legacy systems may require that the compatibility bus be bus 0, which is not allowed to be behind another bridge. Therefore, another approach can be used which allows the PCI bus that contains the subtractive decode device to be configured as bus 0. In this approach, the LDT to PCI bridge implements a function header rather than a bridge header.

- The bridge performs subtractive decode for transactions which originate on the primary bus, using the Compat bit.
- The bridge claims all transactions that originate on the secondary bus and forwards them to the primary bus, but it does not do this subtractively. This implies that peer-to-peer transactions that are targeted at devices on the secondary bus and sourced on either the secondary or subordinate buses are not supported.

- Any type 1 configuration access targeted at bus 0 and forwarded down this LDT chain by the host would have its Compat bit set. Therefore it will reach this bridge. The bridge will decode the request, recognize that it is targeted at bus 0, and forward a type 0 configuration access to the secondary bus.

E.4 VGA Palette Snooping

The *PCI Local Bus and Bridge Architecture Specifications* define VGA palette snooping. This allows a device on the same bus as the device owning the VGA palette range or a bridge that forwards the VGA palette range, to pick up write data as the access goes by.

No direct support for VGA palette snooping is provided in LDT. It can be supported at a level above the LDT protocol by designating an address range as an alias of the VGA palette range and by having the host bridge generate a posted write to the alias as well as the write to the original address. The snooping device must recognize the aliased write and translate it back to the VGA palette range before forwarding or operating on it. The details of this mechanism are implementation-specific.

F Required LDT Behavior In X86 Platforms

The following specifies most expected behavior of LDT protocol that is specific to x86 platforms. For devices designed for x86 platforms, refer to mandatory requirements in the *LDT System Design Guide for x86 Platforms*.

F.1 Mapping Legacy 8259 (PIC) Interrupts

The PIC is assumed to reside in the Southbridge. When INTR out of the PIC is asserted, the Southbridge generates an LDT interrupt request message as follows: MT=ExtInt, TM=edge, DM=physical, INTRDest=FFh, Vector=00h. The host processes the interrupt request as a non-vectored interrupt as described in section 5.1. The processor which services the interrupt request issues an interrupt acknowledge (RdSized) cycle to the Southbridge. The interrupt vector is returned in the eight least-significant bits of the RdResponse. The 24 most-significant bits are zero. Even when legacy PIC interrupts are configured as level-sensitive, the LDT interrupt request is sent as edge mode to indicate that an LDT EOI is not used. EOI to the legacy PIC is performed as an IO access to the PIC address, not an LDT EOI.

F.2 System Management

The system Southbridge is defined, in part, to include the platform system management logic that controls ACPI-defined and platform-specific system state transitions. The Southbridge and host use LDT system management messages to facilitate system state transitions.

X86-platform Southbridges are required to include BIOS-programmable configuration registers called system management action fields (SMAF). These specify the value for bits[3:1] of the STPCLK assertion LDT system management message sent from the Southbridge to the host, based on the system state transition being executed. The Southbridge is required to provide separate BIOS-programmable SMAF registers for (1) each ACPI-defined state (as well as throttling) supported by the Southbridge and (2) host-initiated VID/FID changes. These registers are to be programmed by BIOS after boot, prior to any system state transitions from the fully operational state.

X86-platform hosts may alter their behavior for system state transitions based on the SMAF values transmitted with STPCLK assertion LDT system management messages.

The Southbridge is required to control LDTSTOP#, in support of VID/FID change. It may optionally be asserted during other system state transitions and LDT link width or frequency changes as well. No other devices are allowed to control LDTSTOP#.

All system state transitions and LDT link width or frequency changes forced by LDTSTOP# follow this sequence:

1. The sequence starts with one of the following three methods: (1) the host accesses a Southbridge register (as is the case with ACPI-defined system and CPU sleep state transitions), (2) the host sends a VID/FID change LDT system management cycle to the Southbridge, or (3) the Southbridge logic initiates the sequence without an LDT transaction (as is the case with throttling).
2. The Southbridge responds by sending a STPCLK assertion LDT system management message to the host with UnitID matching the UnitID of the response to the host access from step 1, if a response is required. Bits[3:1] of this message contain the SMAF value associated with the system state transition being executed.
3. After the STPCLK assertion message is sent to the host, the Southbridge may send the response to the initiating transaction from step 1, if a response is required, with the PassPW bit cleared. Such responses are required to follow the STPCLK assertion LDT system management message to guarantee that the host does not execute any additional instructions after the initiating command of step 1, as is required by some operating systems.
4. The host is required to respond to the STPCLK assertion LDT system management message with a STOP_GRANT LDT system management message. This is intended to indicate that the host is ready for the next step in the state transition. Note: the Southbridge is required to wait for the STOP_GRANT LDT system management message prior to sending a STPCLK deassertion LDT system management message. Note: the following steps assume that RESET# is not asserted as part of the system power state transition; if RESET# is asserted, it must be asserted after the STOP_GRANT LDT system management message is received by the Southbridge; the resume that occurs after the reset are as specified in section 12.
5. The Southbridge may assert LDTSTOP# after the STOP_GRANT LDT system management message is received, based on the system state transition being executed. If a VID/FID transition is being executed, then the Southbridge is required to assert LDTSTOP#.
6. If LDTSTOP# was asserted, then it may be deasserted, as required by the system management logic.
7. After LDTSTOP# is deasserted, the Southbridge is required to send the STPCLK deassertion LDT system management message to the host in order for the host to resume to the fully operational state.

F.2.1 VID/FID changes.

The Southbridge is required to support VID/FID changes as follows:

- Execute the system state transition specified above in response to the VID/FID message from the host.
- Assert LDTSTOP# as described in the above sequence.
- Include a BIOS-programmable configuration register that specifies the LDTSTOP# assertion time associated with VID/FID change system state transitions. Values ranging from 1 microsecond to 100 microseconds are recommended.

F.2.2 Throttling.

Throttling differs from most system state transitions in that the Southbridge sends STPCLK assertion messages to the host without direct initiating messages. Because of this, the possibility of a deadlock exists when the host initiates a system state transition simultaneously with a STPCLK assertion message from the Southbridge. Therefore, to avoid this possibility, the following Southbridge requirements exist:

- If a STPCLK assertion message is sent from the Southbridge for throttling and then a system state transition is initiated via a non-posted access from the host to the Southbridge prior to the STOP_GRANT message for throttling, then the Southbridge is required to send another STPCLK

assertion message to the host with the SMAF field as for programmed for the host-initiated system state transition. The response to the host access must then follow.

- If a STPCLK assertion message is sent from the Southbridge for throttling and then a system state transition is initiated via a posted access from the host to the Southbridge (such as the VID/FID system management cycle), then the Southbridge is required (1) wait for the STOP_GRANT system management message from the host, (2) send a STPCLK deassertion message, and (3) send the STPCLK assertion message to the host with the SMAF field as programmed for the host-initiated system state transition.

There is no deadlock possibility when roughly coincident throttling STPCLK assertion messages occur with interrupt requests. They are naturally resolved as follows:

- If a STPCLK assertion message is sent from the Southbridge for throttling simultaneous with a host-initiated non-posted command that results in an interrupt request (e.g. SMI), then the Southbridge sends the interrupt request to the host followed by the response to the non-posted command. The host is required to send the STOP_GRANT system management message after it receives the response.
- If an asynchronous interrupt request (not initiated by a host non-posted request) is received by the host after the STPCLK assertion message, then the interrupt request is accepted by the host regardless of whether the STOP_GRANT system management message has been sent. However, the host might not act on the interrupt request until the STPCLK deassertion message is received by the host.

F.2.3 C3 System State Transitions and LDTREQ#.

It is possible that LDTSTOP# will be asserted during system state transitions to ACPI-defined C3 (this is only expected on battery-powered platforms). A Southbridge on such a platform is required to deassert LDTSTOP# when any devices require use of LDT. It is recommended that this be accomplished through a signal specified here called LDTREQ#.

LDTREQ# is an open-drain signal connected to all LDT devices peripherals on the platform. It can be asserted by any LDT device while LDTSTOP# is asserted to indicate to the Southbridge that an LDT transaction is required somewhere in the system. The Southbridge responds by deasserting LDTSTOP# and transitioning the system to the full on state.

It is expected that LDTREQ# be used to control the ACPI-defined BM_STS bit in the Southbridge. As a result, LDTREQ# is required to be asserted whenever a device requires use of LDT, regardless of the host state or whether LDTSTOP# is asserted.

F.2.4 SMI and STPCLK.

The system Southbridge is the only device that is allowed to generate SMI interrupts and STPCLK system management messages over LDT. Since both of these messages replace legacy signals, they have special ordering requirements to remain compatible with legacy behavior. In legacy systems, both of these signals have the following behavior:

1. The host causes the signal to be asserted with an instruction that (1) requires a response and (2) allows for no further instruction execution until the response is received.
2. The host detects the assertion of the signal prior to the response.
3. After the response, the host responds to the signal (by taking the SMI interrupt or initiating the STPCLK sequence) prior to executing any more instructions.

Thus, to replicate this behavior, the following requirements exist:

- The Southbridge may generate SMI or STPCLK messages in response to host-initiated transactions. If the host-initiated transaction requires a response, then the response is required to follow the SMI or STPCLK message upstream.
- The UnitID of the SMI or STPCLK message must match the UnitID of the response, or the upstream ordering between the two is not guaranteed.

- In order to gurantee that the response to the host does not pass the SMI or STPCLK message, the PassPW bit in the response must be clear, even if it is a TgtDone.
- As long as the SMI or STPCLK message is received prior to the response to the initiating instruction, the host is required to guarantee that it execute no more instructions beyond the initiating instruction, before it responds to the SMI or STPCLK message.

The host bridge responds to SMI with the SMIACK assertion LDT system management message down all LDT chains. The SMIACK assertion message is required to represent the system state of all the processing elements behind the host bridge. Therefore, the host responds with a single SMIACK assertion message after the SMI interrupt is received. However, as processing elements behind the host bridge exit the SMIACK state, multiple SMIACK deassertion messages may be sent downstream from the host bridge.

The Southbridge is required to send no more than one SMI interrupt request message before receiving the SMIACK assert system management message. After the SMIACK assert system management message is received by the Southbridge, it may send another SMI to the host.

F.2.5 Default State Of Virtual Wires.

It is required that the state of the virtual wires in the Southbridge and the host match after reset. The default state for all virtual wires, including all interrupts, IGNNE, A20M, FERR, STPCLK, and SMIACK, is deasserted.

F.3 Initialization Issues.

Hosts on x86 platforms may not be capable of accepting upstream requests until initialized by software. Therefore, it is required that after deassertion of RESET#, no upstream system management messages, interrupt requests, fences or flushes be generated until enabled by the host. The method used to meet this requirement is outside the scope of this specification. Note that upstream sized read and write requests are also disabled after RESET# by the bus master enable configuration bit.

F.4 AGP Bridge Issues.

Some legacy operating systems require that the location of AGP-specific configuration registers be hardwired as follows:

- The AGP-defined capabilities header must be in bus 0, device 0, function 0.
- The AGP aperture base address register must be at bus 0, device 0, function 0, offset 10h.

Therefore, to meet these requirements, it is recommended that AGP devices be designed as follows:

- The AGP bridge resides on the LDT chain specified to be bus 0.
- The AGP device uses multiple UnitIDs.
- The base UnitID register is programmed to zero at the conclusion of LDT sizing. A different UnitID value must be used during the initialization sequence (See section 12.3).
- The device ID that matches the base UnitID register contains the capabilities header and the AGP aperture base address register (at offset 10h).
- The device ID that is one greater than base UnitID is used for the PCI-to-PCI bridge header that corresponds to the AGP bridge.
- The UnitID that matches the base (zero) is not used for any AGP-initiated I/O streams or responses so that there is no conflict with host-initiated I/O streams or responses. Only the UnitIDs greater than the base are used for I/O streams.

- It is expected that the AGP-defined GART is located in the host. Therefore, the AGP aperture base address register--and any other registers that are located in the AGP device but required by the host--are copied via software into implementation-specific host registers.

In the situation described above, the host's configuration registers should be placed somehere other than device zero, in order to avoid conflicting with the predefined AGP registers.

Note that if legacy OS support is not required, the AGP device's base UnitID register may be programmed to any LDT-compliant value.

G CRC Testing Mode

Writing a 1 to the CRC Start Test bit of the Link Control register (see section 7.5.4.2) causes the transmitter to enter CRC diagnostic mode. The transmitter begins by issuing a Nop packet with its Diag bit set, which instructs the receiver to ignore the CAD and CTL lines for the following 512 bit times in each byte lane, not counting the bit times allotted to CRC stuffing. The transmitter can then drive any pattern it wants on the CAD and CTL lines (other than during CRC stuffing), even to the extent of allowing CTL to change state between arbitrary bit times, with one exception. The test pattern may not contain four consecutive bit times of all 1 bits on any byte lane (CAD and CTL lines), as that would be interpreted by the receiver as a sync packet. How the transmitter decides what to transmit as a test pattern is beyond the scope of this specification. CRC is still generated and checked for the interval, and CRC stuffing occurs normally, but the received data is ignored, and packet generating and tracking state machines are suspended in the state they were in when the diagnostic Nop was received. CRC errors detected during this time will be logged by setting the CRC Error bits, and will be treated as fatal if the CRC Flood Enable bit is set. If the CRC Force Error bit (section 7.5.4.3) is set when the CRC Start Test bit is set, the test pattern will contain at least one CRC error in each active byte lane. When the test interval has completed, and the last CRC covering any part of the test interval has been stuffed, hardware clears the CRC Start Test bit in the transmitter. Packet transmission resumes from the suspended state, which may be in the middle of a data packet. This test mode should not be used unless both sides of the link indicate support for it in bit 2 of the feature capability register as defined in section 7.5.9.3.

H Doubleword-Based Data Buffer Flow Control

LDT provides an operating mode in which posted request data, nonposted request data, and response data buffers are flow controlled with doubleword granularity. In this mode the command packets (posted requests, nonposted requests and responses) are still flow controlled using packet size granularity. All LDT components must support the 64-byte granular flow control mode as previously described. Further, 64-byte granular flow control is the default operation for all devices after cold reset (see section 12.1 for the definition of cold reset). Initialization firmware can determine the capability of components on either side of a link, and if they support doubleword-based flow control, program them to operate in that mode. Switching between flow control modes requires cycling through a software-initiated warm reset. An LDTSTOP# disconnect sequence cannot be used to switch between flow control modes because flow control buffer state must be kept consistent across LDTSTOP# disconnects.

The diagram below shows the Nop packet format for doubleword-based flow control. In this mode the two-bit data buffer flow control fields previously described are interpreted as the upper two bits of a five-bit flow control field. There are three five-bit fields in total, each corresponding to one of the three virtual data channels. Each five-bit field indicates to the transmitter that the receiver is freeing from zero to 31 doublewords of data buffer within a channel. The byte mask doubleword for sized byte writes is included as data in the doubleword-based flow control calculation by both the transmitter and the receiver.

Bit Time	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Isoc	DisCon	Cmd[5:0]					
1	ResponseData[4:3]		Response[1:0]		PostData[4:3]		PostCmd[1:0]	

Bit Time	7	6	5	4	3	2	1	0
2	0	Diag	Rsv	Resp Data [2]	NonPostData[4:3]		NonPostCmd[1:0]	
3	ResponseData[1:0]		PostData[2:0]			NonPostData[2:0]		

As in packet-based data buffer flow control mode, if a transmitter receives more increments than it can keep track of, it must not allow its counter to wrap, but must discard the extras. This has the effect that the link will use the maximum amount of buffer storage that both the transmitter and receiver can support. All transmitter counters must be a minimum of six bits wide, allowing up to 63 double words of buffer storage to be tracked without loss.

Doubleword-based flow control is expected to be deployed only in special circumstances where large block sized, high-performance transfers are not important to the operation of the LDT component within the system. All LDT components must support 64-byte flow control mode, and are encouraged to implement a large enough 64-byte buffer pool to fully utilize the LDT links in the system applications envisioned for that component.